ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Evaluación del dispositivo *Raspberry Pi* como elemento de despliegue de servicios en el marco de las *Smart Grids*

AUTOR: Luis Manuel Moreno Rodríguez

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR (o Director en su caso): Rubén de Diego Martínez

DEPARTAMENTO: DIATEL (Departamento de Ingeniería y Arquitecturas Telemáticas)

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Martina Eckert

VOCAL: Rubén de Diego Martínez

SECRETARIO: Vicente Hernández Díaz**.**

Fecha de lectura:

Calificación:

El Secretario,

Agradecimientos

En primer lugar, agradecer a mis amigas Nora, Paula, Raquel, Miriam y Almudena por haberme apoyado cuando lo necesitaba, y en especial a mi amiga Verónica, que siempre ha estado ahí.

A mis amigos Carlos, Luis, Óscar, Alberto y Daniel, por haberme acompañado durante la carrera, habiendo compartido momentos muy buenos, y otros no tan buenos.

A mis amigos Miguel, Félix, Álvaro y Jesús Javier, que me han distraído cuando lo necesitaba.

También quiero agradecer toda la ayuda recibida por mi tutor, Rubén de Diego, por su paciencia y ayuda a la hora de redactar esta memoria, agradecimiento que hago extensivo a los miembros del GRyS-CITSEM, que han permitido que desarrolle este proyecto.

Por último, quiero dedicar un agradecimiento especial a mis padres, que siempre me han impulsado a mejorar y que me han servido de apoyo en los momentos duros de la carrera.

Resumen

La tendencia actual de las redes de telecomunicaciones conduce a pensar en un futuro basado en el concepto emergente de las *Smart Cities*¸ que tienen como objetivo el desarrollo urbano basado en un modelo de sostenibilidad que responda a las necesidades crecientes de las ciudades.

Dentro de las *Smart Cities* podemos incluir el concepto de *Smart Grid*, el cual está referido a sistemas de administración y producción de energía eficientes, que permitan un sistema energético sostenible, y que den cabida a las fuentes de energía renovables. Sistemas de este tipo se muestran a los usuarios como un conjunto de servicios con los que interactuar sin ser tan sólo un mero cliente, sino un agente más del entorno energético.

Por otro lado, los sistemas de software distribuidos son cada vez más comunes en una infraestructura de telecomunicaciones cada vez más extensa y con más capacidades. Dentro de este ámbito tecnológico, las arquitecturas orientadas a servicios han crecido exponencialmente sobre todo en el sector empresarial. Con sistemas basados en estas arquitecturas, se pueden ofrecer a empresas y usuarios sistemas software basados en el concepto de servicio.

Con la progresión del hardware actual, la miniaturización de los equipos es cada vez mayor, sin renunciar por ello a la potencia que podemos encontrar en sistemas de mayor tamaño. Un ejemplo es el dispositivo *Raspberry Pi*, que contiene un ordenador plenamente funcional contenido en el tamaño de una cajetilla de tabaco, y con un coste muy reducido.

En este proyecto se pretenden aunar los tres conceptos expuestos. De esta forma, se busca utilizar el dispositivo *Raspberry Pi* como elemento de despliegue integrado en una arquitectura de *Smart Grid* orientada a servicios. En los trabajos realizados se ha utilizado la propuesta definida por el proyecto de I+D europeo *e-GOTHAM*, con cuya infraestructura se ha tenido ocasión de realizar diferentes pruebas de las descritas en esta memoria. Aunque esta arquitectura está orientada a la creación de una *Smart Grid*, lo experimentado en este PFG podría encajar en otro tipo de aplicaciones.

Dentro del estudio sobre las soluciones software actuales, se ha trabajado en la evaluación de la posibilidad de instalar un *Enterprise Service Bus* en el *Raspberry Pi* y en la optimización de la citada instalación. Una vez conseguida una instalación operativa, se ha desarrollado un controlador de un dispositivo físico (sensor/actuador), denominado *Dispositivo Lógico*, a modo de prueba de la viabilidad del uso del *Raspberry Pi* para actuar como elemento en el que instalar aplicaciones en entornos de *Smart Grid* o *Smart Home*.

El éxito logrado con esta experimentación refuerza la idea de considerar al *Raspberry Pi*, como un importante elemento a tener en cuenta para el despliegue de servicios de *Smart Cities* o incluso en otros ámbitos tecnológicos.

Abstract

The current trend of telecommunication networks lead to think in a future based on the emerging concept of *Smart Cities*, whose objective is to ensure the urban development based on a sustainable model to respond the new necessities of the cities.

Within the Smart cites we can include the concept of *Smart Grid*, which is based on management systems and efficient energy production, allowing a sustainable energy producing system, and that includes renewable energy sources. Systems of this type are shown to users as a set of services that allow users to interact with the system not only as a single customer, but also as other energy environment agent.

Furthermore, distributed software systems are increasingly common in a telecommunications infrastructure more extensive and with more capabilities. Within this area of technology, service-oriented architectures have grown exponentially especially in the business sector. With systems based on these architectures, can be offered to businesses and users software systems based on the concept of service.

With the progression of the actual hardware, the miniaturization of computers is increasing, without sacrificing the power of larger systems. An example is the *Raspberry Pi*, which contains a fully functional computer contained in the size of a pack of cigarettes, and with a very low cost.

This PFG (*Proyecto Fin de Grado*) tries to combine the three concepts presented. Thus, it is intended to use the Raspberry Pi device as a deployment element integrated into a service oriented Smart Grid architecture. In this PFG, the one proposed in the European R&D e-GOTHAM project has been observed. In addition several tests described herein have been carried out using the infrastructure of that project. Although this architecture is oriented to the creation of a *Smart Grid*, the experiences reported in this document could fit into other applications.

Within the study on current software solutions, it have been working on assessing the possibility of installing an *Enterprise Service Bus* in the *Raspberry Pi* and optimizing that facility. Having achieved an operating installation, it has been developed a driver for a physical device (sensor / actuator), called logical device, for testing the feasibility of using the *Raspberry Pi* to act as an element in which to install applications in *Smart Grid* and *Smart Home* Environments.

The success of this experiment reinforces the idea of considering the *Raspberry P*i as an important element to take into account in the deployment of *Smart Cities* services or even in other technological fields.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de gráficos

Acrónimos

AGPL: Affero General Public License.
AMQP: Advanced Message Queuing Protocol.
AMS: Advanced Metering System.
API: Application Programming Interface.
ARM: Advanced RISC Machines.
AOP: Aspect Oriented Programming.
ASF: Apache Software Foundation.
ASP: Active Server Pages.
B2B: Businnes-2-Business.
BPA: Business Process Automation.
BPM: Business Process Management.
CDDL: Common Development and Distribution License.
CORBA: Common Object Request *Broker* Architecture.
CPAL: Common Public Attribution License.
DM: Dynamic Modules.
DSP: Digital Signal Processor.
EAI: Enterprise Application Integration.
EIP: Enterprise Integration Patterns.
EMS: Energy Monitoring Systems.
ESB: Enterprise Service Bus.
FAT: File Allocation Table.
FIFO: First Input, First Output.
FTP: File Transfer Protocol.
GiB: Gigabyte.
GPIO: General Purpose Input/Output.
GPL: General Public License.
GPU: Graphics Processing Unit.
HAL: Hardware Abstraction Layer.
HDMI: High Definition Multimedia Interface.
HTML: Hypertext Markup Language.
HTTP: Hypertext Transfer Protocol.
HVAC: Heating, Ventilation and Air Conditioning.
I/O: Input/Output.
IoT: Internet of Things.
JAR: Java Archive.
JAX: Java API for XML.
JBI: Java Business Integration.
JDK: Java Development Kit.
JMS: *Java Message Service*.
JNDI: Java Naming and Directory Interface.
JRE: Java Runtime Environment.
JVM: Java Virtual Machine.
KiB: Kilobyte.
LGPL: Lesser General Public License.

MiB: Megabyte.
MMC: Multi Media Card.
MOM: Message Oriented Middleware.
MQ: Message Queuing.
MTOM: Message Transmission Optimization Mechanism.
NTSC: National Television System Committee.
OS: Operating System.
OSGi: Open Services Gateway Initiative.
PAL: Phase Alternating Line.
PC: Personal Computer.
PHP: Hypertext Preprocessor.
POM: Project Object Model.
PTP: Point to Point.
PVP: Precio de Venta al Público.
RAM: Random Access Memory.
REI: Red eléctrica inteligente.
RES: Renewable Energy Sources.
REST: Representational State Transfer.
RISC: Reduced Instruction Set Computer.
RJ45: Registered Jack 45.
RPC: Remote Procedure Call.
RRSHB: Resource Representation SOAP Header Block.
SATA: Serial Advanced Technology Attachment.
SBC: Single-Board Computer.
SCTP: Stream Control Transmission Protocol.
SD: Secure Digital.
SDHC: Secure Digital High Capacity.
SDRAM: Synchronous Dynamic Random Access Memory.
SMTP: Simple Mail Transfer Protocol.
SOA: Service Oriented Architectures.
SOAP: Simple Object Access Protocol.
SOC: System-On-Chip.
STOMP: Stream Text Oriented Messaging Protocol.
TCP/IP: Transmission Control Protocol / Internet Protocol.
URI: Uniform Resource Identifier.
USB: Universal Serial Bus.
UTF: Unicode Transformation Format.
VM: Virtual Machine.
WADL: Web Application Description Language.
WiFi: Wireless Fidelity.
WS: Web Services.
WSDL: Web Services Description Language.
WSN: Wireless Sensor Networks.
XML: eXtensible Markup Language.
XSD: XMSL Schema Definition,
XSL: eXtensible Stylesheet Language Family.
XSLT: eXtensible Stylesheet Language Family Transformations.

1. Introducción

El aumento progresivo de la población en las ciudades hace que el consumo energético se dispare. El consumo de energía eléctrica es uno de los factores más importantes de consumo dentro de las ciudades, y con el aumento de dispositivos tanto eléctricos como electrónicos en los hogares, este consumo es cada vez mayor.

Por otro lado, el aumento del consumo de energía eléctrica repercute en los usuarios aumentando el coste de contratación de servicios.

Si bien el abastecimiento de energía eléctrica es adecuado en los tiempos actuales, no posee un modelo de sostenibilidad a largo plazo, ni resulta eficiente. Entre algunas de las causas [1] [2], se puede encontrar que:

- La producción de energía no se ajusta a la demanda en cada instante, si no que se produce una cantidad de energía determinada de acuerdo a diferentes parámetros, como la zona abastecida, o el horario.
- El desperdicio de energía afecta al ecosistema, ya que no se establecen medidas para aprovechar dicha energía. Además, al producirla se aumentan las emisiones contaminantes.
- Las energías renovables, si bien son cada vez más utilizadas, todavía no se incluyen como una parte importante del sistema.
- Los usuarios no son capaces de controlar de forma efectiva el consumo, ya que no disponen de herramientas para ello.

A raíz de todo esto, surge un nuevo concepto de red eléctrica, denominada *Smart Grid* [3] [4]. Una *Smart Grid* permite entre otras cosas:

- Ajustar dinámicamente la producción y el consumo de energía.
- Dotar de cierta inteligencia a la red eléctrica, de tal forma que los usuarios sean un agente más de la red.
- Integrar las nuevas energías renovables, para mejorar el medioambiente.

El proyecto descrito en esta memoria está relacionado con el campo de las *Smart Grid* a través del proyecto europeo de I+D *e-GOTHAM*, en cuya infraestructura se apoya para la realización de los objetivos planteados.

1.1. Objetivos

El objetivo principal de este proyecto consiste en evaluar la utilización de un ordenador de tamaño reducido, llamado *Raspberry Pi*, como elemento para el despliegue de servicios.

Para llevar a cabo este objetivo, se realizará un estudio en tres fases que se ilustran en la Figura 1 en las cuales se buscarán conseguir los objetivos parciales que se indican a continuación.

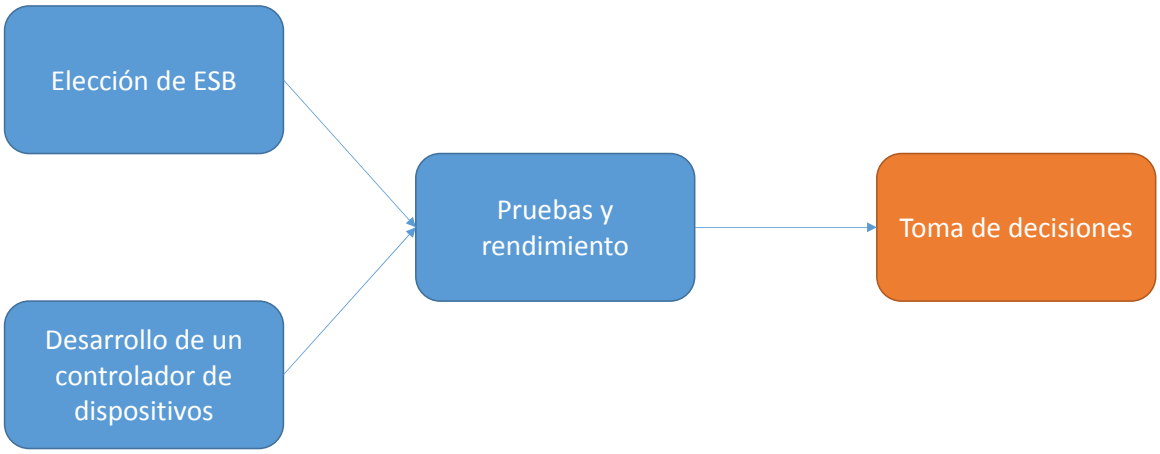

Figura 1. Fases de desarrollo del proyecto.

Para la fase 1 se realizará un estudio completo de las soluciones ESB actuales, que concluirá con la elección de uno de estos programas como elemento de implementación SOA en el *Raspberry Pi*. Se busca conseguir:

- Realizar un estudio y una comparativa de los ESB más utilizados.
- Encontrar un candidato válido para su instalación en el *Raspberry Pi.*

Para la fase 2 se desarrollará un controlador de dispositivos que permita la realización de pruebas en el entorno de este PFG. Se buscará:

- Crear un controlador de dispositivos totalmente funcional.
- Implementar sistemas en el controlador para la realización de pruebas de rendimiento.

Para la fase 3 se realizarán las pruebas de rendimiento del controlador de dispositivos instalado en el ESB elegido en la fase 1. Mediante estas pruebas se buscará:

- Obtener estadísticas y datos reales de rendimiento del *Raspberry Pi*.
- Realizar la valoración final sobre el dispositivo *Raspberry Pi.*

1.2. Estructura del documento

Los principales contenidos teóricos y el marco tecnológico del proyecto se abordarán en el capítulo 2 de este documento, en el que se mostrarán:

- Las funciones y estructura de una *Smart Grid*.
- Las principales tecnologías en el campo de las arquitecturas orientadas a servicios.

El entorno en el que se desarrollará el proyecto se detallará en el capítulo 3, centrándose en las especificaciones del *Raspberry Pi,* y la explicación de la finalidad y el funcionamiento del controlador de dispositivos.

La fase 1 está descrita en el capítulo 4, y en ella se expone el estudio de los ESB realizado, así como las comparativas llevadas a cabo. Como conclusión de esta fase se incluirá la solución elegida, junto con su descripción en detalle.

La fase 2 está descrita en el capítulo 5, y en él se detalla el proceso de desarrollo de un controlador de dispositivos físicos, y su despliegue de acuerdo a la arquitectura de *Smart Grid* desarrollada en el proyecto europeo *e-GOTHAM*.

La fase 3 está descrita en el capítulo 6, y en ella se podrán encontrar los resultados de las pruebas realizadas, así como la valoración de los resultados obtenidos en cada una de estas pruebas. Por último, se encuentra la valoración final de cumplimiento del objetivo principal de este PFG.

En el capítulo 7 de esta memoria se encontrarán las conclusiones generales del Proyecto Fin de Grado y se propondrán una serie de trabajos futuros con los que continuar el trabajo desarrollado.

Adicionalmente se entrega como separata de esta memoria un Manual de Usuario de la aplicación,

2. Marco tecnológico.

2.1. *Smart Grid*

La *Smart Grid*, o Red Eléctrica Inteligente (REI) es aquella red inteligente capaz de integrar de forma eficiente el comportamiento y las acciones de todos los usuarios y agentes que conforman la red eléctrica, para lograr un sistema de energía que cumple con una serie de objetivos. Estos objetivos [4] se recogen a continuación:

- Conseguir equilibrar de forma dinámica la producción al consumo de energía por parte de los usuarios, economizando la red, y consiguiendo una alta eficiencia.
- Integrar eficazmente las nuevas energías renovables.
- Mejorar la transmisión de la energía a los usuarios.
- Reducir el consumo por parte de los usuarios, que son capaces de controlar dicho consumo de una forma dinámica.
- Reducir las emisiones dañinas con el uso de nuevas energías *verdes*, que contaminan menos el planeta.

Este tipo de redes son cada vez más utilizadas, siendo una parte importante dentro del concepto de *Smart Living*, y engloba soluciones de *Smart Home* para los usuarios.

Parte de las funcionalidades desarrolladas por una *Smart Grid* [3] son las siguientes:

- Conseguir una transmisión más eficiente de la energía.
- Conseguir una mayor robustez en la red frente a incidencias, que en el caso habitual podrían significar el corte de la corriente a los usuarios.
- Reducir los costes de operación, mantenimiento y gestión de las redes eléctricas, y los costes para los usuarios.
- Reducir la demanda de energía, gracias al equilibrio entre producción y consumo.
- Permitir integrar grandes sistemas de energía renovable, como parques eólicos.
- Permitir la integración de sistemas de autosuficiencia energética de los usuarios.
- Aumentar la seguridad de la red eléctrica.

Las principales características de una red de este tipo [4] son las mencionadas a continuación:

- **Flexibilidad y seguridad:** son muy adaptables a los cambios del sistema. Gracias a una interacción bidireccional del usuario y la red es más fácil establecer los cambios que ocurren al poseer información en tiempo real. Esto también aumenta la seguridad de la red eléctrica.

- **Eficiencia:** al ser una red inteligente, la distribución se puede encaminar dependiendo de las necesidades, minimizando así la infraestructura necesaria.

- **Escalabilidad:** la inclusión de nuevos servicios y la expansión geográfica de la infraestructura de distribución se favorecen en estas redes.

- **Sostenibilidad:** se aboga por la inclusión de las energías renovables como fuente principal para la generación de la energía.

2.1.1. Soluciones actuales de *Smart Grid*

Actualmente, y debido al impulso de conceptos como *Smart Living* y *Smart Home*, se encuentran muchas soluciones para la creación de *Smart Grids*. Encontramos multitud de empresas del sector tecnológico, la mayor parte de ellas de una importancia relevante dentro de este sector, que desarrollan soluciones de este tipo, como por ejemplo Siemens [5] o Hewlett Packard [6].

En el documento *Smart Grid Projects Outlook 2014* [7] se podrá encontrar una buena recopilación y descripción de los actuales proyectos de I+D+i europeos relacionados con la *Smart Grid*. Se incluye a continuación una breve descripción de dos proyectos en los cuales participa la Universidad Politécnica de Madrid.

e-GOTHAM

e-GOTHAM se trata de un proyecto de investigación y desarrollo europeo situado en el programa Artemis, que define la agenda de desarrollos tecnológicos en el campo de la computación de sistemas. Este proyecto está dirigido por Inabensa, y en él participan otras diecisiete entidades europeas.

Dicho proyecto promueve una solución de red eléctrica inteligente basada en el concepto de *microgrids*, de tal forma que la red total se divide en subredes más pequeñas que funcionan como pequeñas *Smart Grid*. Se puede ver este concepto en la Figura 2, en la cual se muestran diferentes tipos de *microgrids* conectadas entre sí.

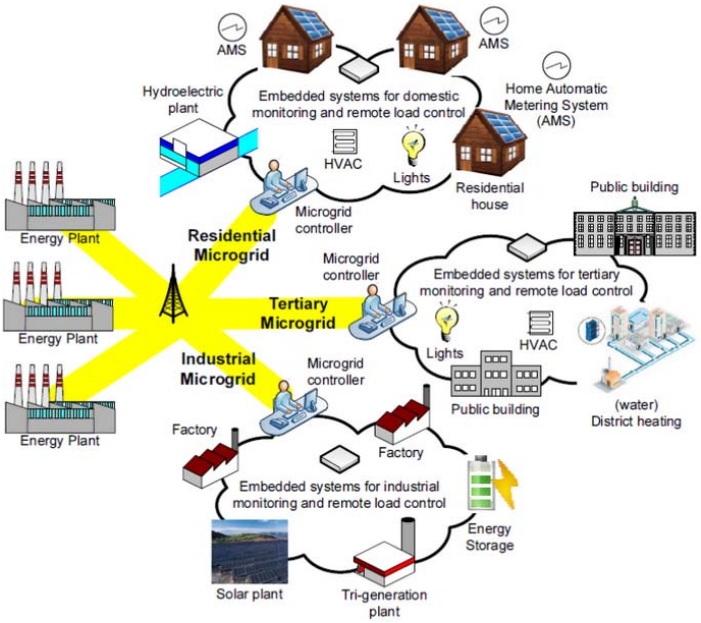

Figura 2. Estructura de Smart Grid propuesta por e-GOTHAM[1].

[1]Fuente: http://www.e-gotham.eu/images/e-gotham%201.png

Pretende implementar un nuevo modelo de demanda agregada de energía, que cumpla con los siguientes objetivos [8]:

- Integrar de forma efectiva nuevas fuentes de energía renovables.
- Incrementar la eficiencia en el mantenimiento y la gestión de la red mediante el equilibrio dinámico entre demanda y consumo de energía.
- Reducir las emisiones de carbono, dando para ello prioridad a fuentes de energía *verdes*.
- Crear conciencia de la importancia de reducir el consumo de energía, implementando para ello productos y servicios a los usuarios con los que puedan administrar su consumo de energía.
- Simular el desarrollo de un mercado puntero para tecnologías de energía eficientes, creando nuevos modelos de negocio.

I3RES

I3RES es un proyecto europeo en cooperación liderado por Inabensa, en el que participan 8 entidades europeas, que ha sido financiado por el Séptimo Programa Marco de la Comisión Europea. Este es el principal programa europeo de investigación, que tiene como objetivos estratégicos principales reforzar la base científica y tecnológica de la industria europea y favorecer su competitividad internacional, promoviendo una investigación que respalde las políticas comunitarias.

I3RES es un proyecto que se basa en la gestión inteligente de la red eléctrica. El objetivo principal de I3RES es desarrollar una herramienta de gestión para la red de distribución, apuntalada por los siguientes conceptos [9]:

- Un sistema de monitorización que integra información de sistemas ya instalados (por ejemplo, SCADA, EMS (*Energy Monitoring Systems*) y medidores inteligentes);
- Previsión de la producción de energía y los algoritmos de gestión de redes que ayudan a la empresa de distribución en la gestión de la producción RES (*Renewable Energy Sources*, Fuentes de Energía Renovables) distribuido masivamente y la producción RES a gran escala dentro de la red de distribución.
- Minería de datos e inteligencia artificial para analizar la demanda de energía y la producción en la red de distribución de los consumidores.

2.2. Arquitecturas orientadas a servicios

SOA (*Service Oriented Architectures*, o arquitecturas orientadas a servicios) describe un paradigma de arquitectura que permite desarrollar sistemas software distribuidos. Este tipo de arquitecturas se basan en el concepto de servicio, definido en el libro *Service-Oriented Architecture. Concepts, Technology and Design*, escrito por Thomas Erls [10] como:

"*Una función sin estado, auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida*."

Para el despliegue de los servicios, se realiza un contrato de servicios estandarizado, en el cual se define la descripción cada uno de estos servicios. Con este contrato, un servicio permite abstraer la lógica de negocio. Algunas de las características básicas de los servicios son [10]:

- Proporcionan un débil acoplamiento entre las aplicaciones, ya que cada servicio es independiente de los demás.
- Incentiva la reutilización de componentes software, por lo que es fácil integrar servicios legados con otros nuevos desarrollados.
- La ubicación de los servicios es transparente al usuario.

Estos servicios se utilizan en sistemas distribuidos, mediante software específico que permite la realización de aplicaciones que utilizan estos servicios. Están destinadas en su mayoría para ofrecer soluciones para aplicaciones empresariales, y ofrecen algunas características que son importantes para ello, como:

- Gran escalabilidad del sistema.
- Facilidad y flexibilidad en la integración de sistemas tanto nuevos como legados.
- Alineación directa de los procesos de negocio.
- Reducción de costes de implementación.
- Adaptación ágil a cambios

2.2.1. Arquitectura de referencia

Estas arquitecturas deben ser implementadas con una serie de componentes básicos que permitan desarrollar las características principales anteriormente definidas.

Las arquitecturas software que deseen implementar este paradigma deben constar de las siguientes capas de software [11]:

- **Aplicaciones básicas**. Sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad.
- **De exposición de funcionalidades**. Donde las funcionalidades de la capa aplicativa son expuestas en forma de servicios (generalmente como servicios web).
- **De integración de servicios**. Facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración.
- **De composición de procesos**. Que define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio.
- **De entrega**. Donde los servicios son desplegados a los usuarios finales.

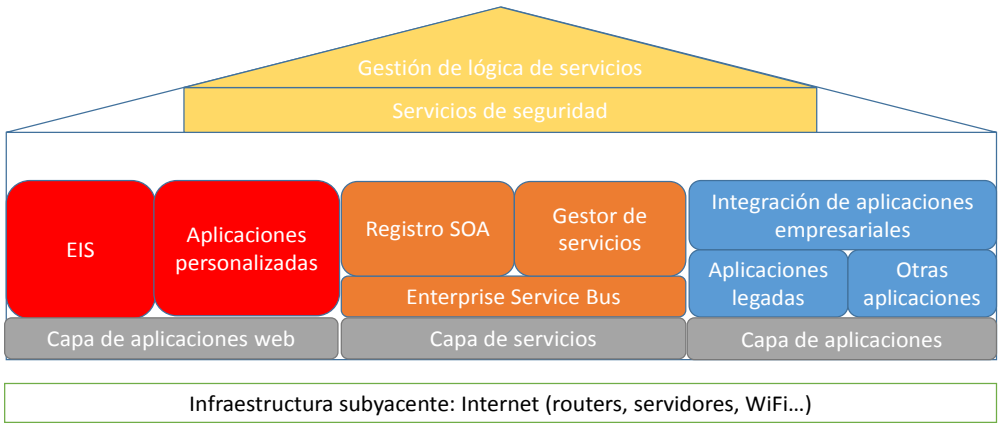

Figura 3. Arquitectura de referencia de SOA.

En la Figura 3 se describe una arquitectura de referencia para este paradigma. En ella se pueden observar cada una de las capas previamente definidas:

- La capa de aplicaciones básicas. Aquí encontraríamos por ejemplo aplicaciones que integren patrones de integración empresariales (EIP, *Enterprise Integration Pattenrns)*, o aplicaciones de otros sistemas legados.
- La capa de exposición de funcionalidades se implementaría en la capa de aplicaciones web. Encontraríamos por ejemplo la definición de *Web Services*.
- En la capa de integración de servicios encontraríamos implementaciones propiamente dichas de la arquitectura, como los ESB (*Enterprise Service Bus*, bus de servicios empresariales).
- Las capas de composición de procesos y entrega estarían englobadas en la gestión de la lógica de los servicios.

2.2.2. Elementos de una arquitectura orientada servicios

En la Figura 4 se pueden observar algunos de los elementos más comunes que forman parte de una arquitectura orientada a servicios.

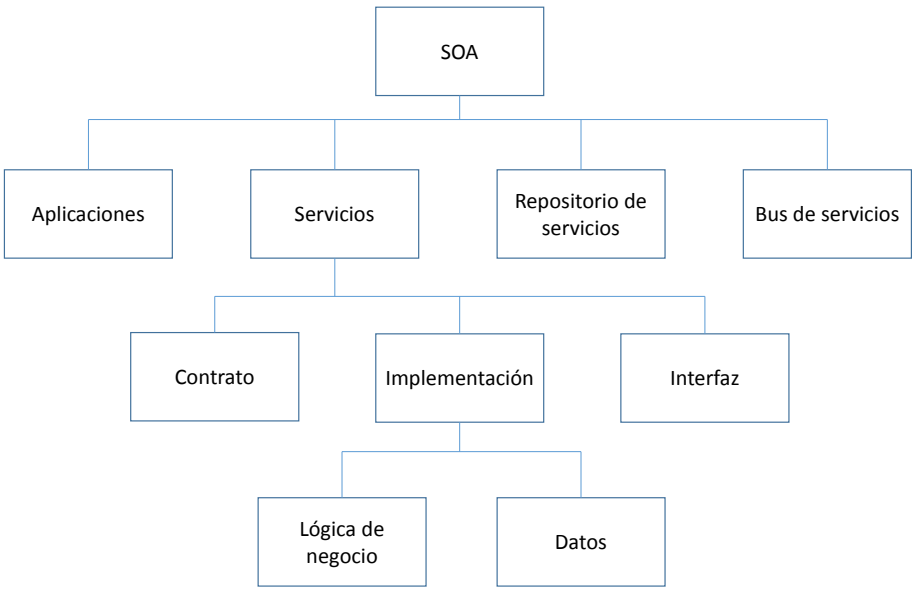

Figura 4. Elementos de una arquitectura orientada a servicios.

El concepto de servicio, que es el elemento más importante del paradigma, se compone de una interfaz de definición para dicho servicio. El servicio dispone de una implementación, que incluye la lógica de negocio y los datos que son manejados por el servicio. Existe un repositorio de servicios, en el que están contenidos todos los servicios administrados por la arquitectura.

El bus de servicios permite a los sistemas intermediar para la ejecución de los servicios mediante las aplicaciones.

2.2.3. Ventajas y desventajas de SOA

La utilización de servicios para el desarrollo de aplicaciones y sistemas distribuidos permite conseguir un nivel de acoplamiento muy bajo de los componentes [12]. Gracias a ello, se favorece la reutilización de componentes, y se aumenta la escalabilidad de los sistemas, ya que no es necesario la reestructuración constante de los componentes de un sistema. Además, el uso de contratos de servicios estandarizados permite conseguir una mayor interoperabilidad en múltiples soportes.

Por otro lado, las aplicaciones apoyadas en servicios están enmarcadas más en el entorno empresarial, para el cual existe más implementación este tipo de arquitecturas. Para su implementación se requiere de un nivel avanzado de conocimiento debido a la complejidad que conlleva.

Por las características propias de este tipo de arquitecturas, no se pueden utilizar para aplicaciones que requieran el intercambio de grandes volúmenes de datos. Además, están pensadas para su desarrollo en entornos de comunicación asíncronos.

2.2.4. Implementaciones

Aunque no es la única, la principal implementación de una arquitectura orientada a servicios se realiza con paquetes software llamados ESB. Los ESB actúan como intermediarios entre sistemas, y establecen un middleware para las comunicaciones entre las aplicaciones. Como principal funcionalidad, proveen una forma de exponer servicios.

Para llevar a cabo la funcionalidad de una arquitectura orientada a servicios, la mayoría de los ESB constan de los siguientes elementos:

- Motor OSGi (*Open Services Gateway Initiative,* o Iniciativa de Pasarela de Servicios Abiertos): OSGi encaja perfectamente con la filosofía de SOA, ya que provee una forma de manejar aplicaciones Java de forma modular, y exponer estos módulos como servicios es una forma de implementar SOA.
- MOM (*Message Oriented Middleware,* software de intermediación orientado a mensajes): permite a las aplicaciones intercambiar información a través de mensajes. Encaja perfectamente con las comunicaciones asíncronas que son las más utilizadas en SOA.
- *Web Service*s, basados en definiciones de interfaces mediante representación de datos utilizando tecnologías XML (*eXtensible Markup Language*).

En los siguientes puntos se explican cada uno de estos componentes, con su tecnología y protocolos asociados.

2.3. OSGi

OSGi son las siglas de *Open Services Gatewaye Initiative*. Es creado en 1999 por la corporación sin ánimo de lucro OSGi Alliance, un consorcio mundial de innovaciones tecnológicas que tiene como objetivo desarrollar un conjunto de definiciones para el proceso de creación de componentes modulares bajo una plataforma Java.

La OSGi Alliance provee especificaciones, referencias de implementación de aplicaciones y servicios, entornos de pruebas y mecanismos de certificación para impulsar una industria tecnológica con un valor añadido. Dicha alianza también promueve la colaboración entre sistemas tanto dentro como fuera de la especificación OSGi para conseguir una evolución del mercado a base de soluciones innovadoras basada en estándares de carácter abierto.

En palabras de la propia alianza, la misión de esta corporación es la consecución de la creación de un *"mercado para middleware universal"* [13].

La tecnología OSGi engloba una serie de especificaciones para definir sistemas de componentes dinámicos en una plataforma Java. Estas especificaciones reducen la complejidad del software, proveyendo para ello una arquitectura modular tanto para grandes sistemas distribuidos como para sistemas más reducidos, e incluso sistemas embebidos.

OSGi facilita la composición de módulos software y aplicaciones, y asegura la gestión remota y la interoperabilidad de aplicaciones sobre una gran cantidad de dispositivos.

2.3.1. Arquitectura OSGi

La arquitectura de OSGi sigue un modelo por capas [14], que se puede ver modelado en la Figura 5.

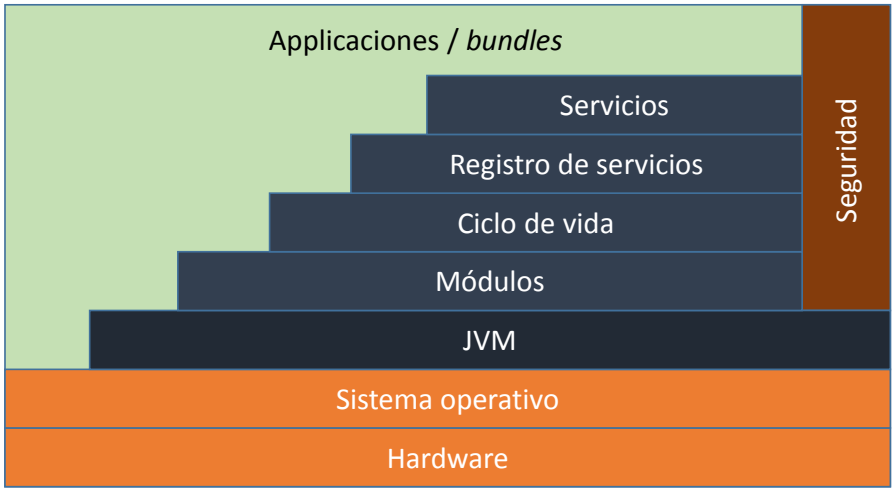

Figura 5. Arquitectura de OSGi.

- ***Bundles* y aplicaciones:** los *bundles* son los componentes OSGi creados por los desarrolladores, y que representan la implementación final. Las aplicaciones utilizan dichos *bundles* para acceder a los servicios.
- **Servicios:** la capa de servicios permite conectar los *bundles* de una forma dinámica, mediante la estructura *publish-find-bind* (publicación-búsqueda-obtención) que permite la importación y exportación de objetos de Java.
- **Ciclo de vida:** esta capa está constituida por una API (*Application Programming Interface* o interfaz de programación de aplicaciones) que permite realizar operaciones sobre los *bundles* dentro de su entorno de ejecución, como iniciar o parar un *bundle.*
- **Módulos:** esta capa define la forma en la cual los *bundles* importan y exportan el código contenido en ellos.

- **Java VM y Sistema operativo:** estas dos capas definen el soporte sobre el que se asienta OSGi. Este soporte está dado por una máquina virtual Java, que opera con el hardware del sistema a través del sistema operativo.
- **Hardware:** esta es la capa física donde se asienta toda la estructura de capas mencionada.

Según la OSGi Alliance [14] los elementos anteriormente definidos para la arquitectura de OSGi son los mostrados a continuación.

Módulos

El principal elemento que hace posible un concepto como el que OSGi intenta transmitir es la modularidad. La modularidad se refiere al uso de componentes separados e independientes entre sí, que actúan en conjunto para el desarrollo de la funcionalidad completa de un sistema. Mediante este concepto, el diseño de software permite crear sistemas con muy bajo acoplamiento, que los vuelven muy escalables, y muy robustos frente a cambios estructurales.

OSGi utiliza el concepto de modularidad mediante el diseño de módulos basados en Java, que definen la lógica de negocio para el código de los *bundles.*

Los *bundles* no son más que una implementación de estos módulos, y están basados en un empaquetamiento de las clases de ejecución del programa, con un documento que define las características y atributos de la aplicación, así como el entorno de ejecución. Esto se puede observar en la Figura 6, en la que se puede ver la estructura de un *bundle,* y su interactuación con la capa de servicios para la importación y exportación dinámica de objetos Java.

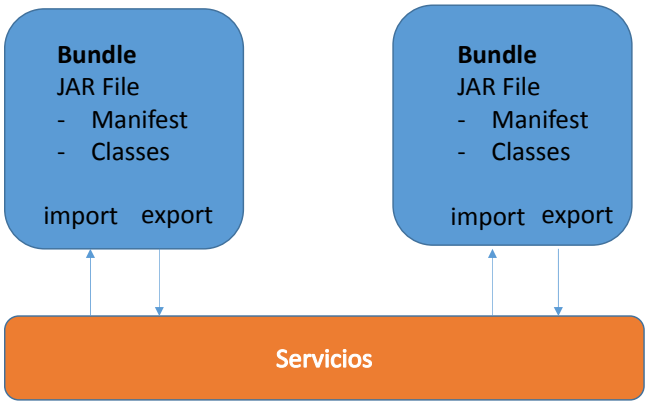

Figura 6. Estructura de los bundles, y su interactuación con los servicios.

Los *bundles* se ejecutan por tanto como módulos, que exportan e importan servicios. La capa de ciclo de vida define el estado de los *bundles* del sistema, como se indica en la Figura 7.

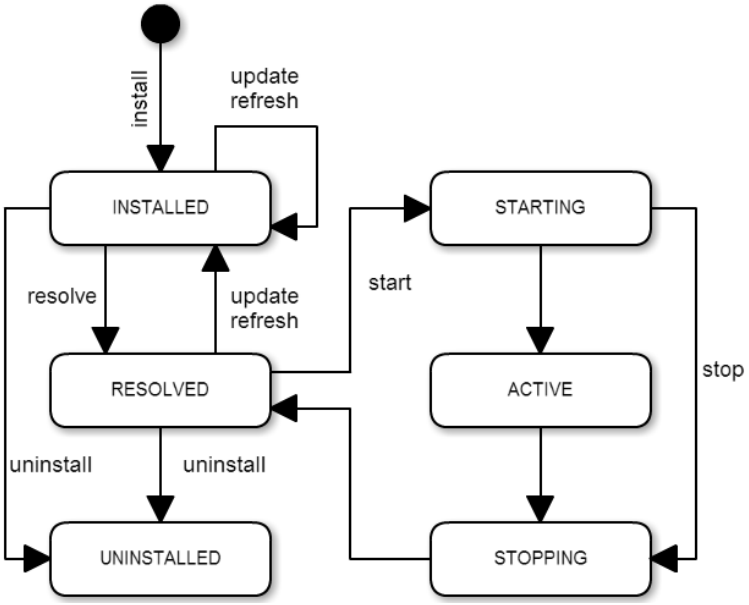

Figura 7. Estados y transiciones entre ellos de un bundle.

Un *bundle*, como ya se ha dicho, puede tener los estados definidos en la figura, que son los mostrados a continuación [15]:

- **INSTALLED** – El *bundle* ha sido instalado correctamente.
- **RESOLVED** – Todas las clases Java que el *bundle* necesita están disponibles. Este estado indica que el *bundle* está listo para ser arrancado o que el *bundle* ha sido detenido.
- **STARTING** – El *bundle* está arrancando.
- **ACTIVE** – El *bundle* ha sido correctamente activado y está en funcionamiento.
- **STOPPING** – El *bundle* ha sido detenido.
- **UNINSTALLED** – El *bundle* ha sido desinstalado. No se puede mover a otro estado.

El cambio entre los diferentes estados de un *bundle* dentro de OSGi es el especificado en la Figura 7.

Servicios

En Java, escribir modelos colaborativos de aplicaciones presenta una cierta dificultad. La forma en la que tradicionalmente se soluciona esto es el uso de clases *factoría* en Java, para la importación dinámica de clases y la conciencia de estados de los objetos.

Sin embargo, se trata de una solución limitada, ya que sólo se puede conocer el estado de los objetos, pero no su disponibilidad. Además, no hay una propuesta centralizada de implementación de este tipo de soluciones.

La solución que propone OSGi se basa en un modelo de servicios. Mediante este modelo, un *bundle* puede exportar un objeto de Java como un servicio, implementado en una interfaz, que se expone a los demás *bundles* mediante una referencia a dicho objeto. La estructura en la que los *bundles* exportan e importan los servicios es la mostrada en la Figura 8.

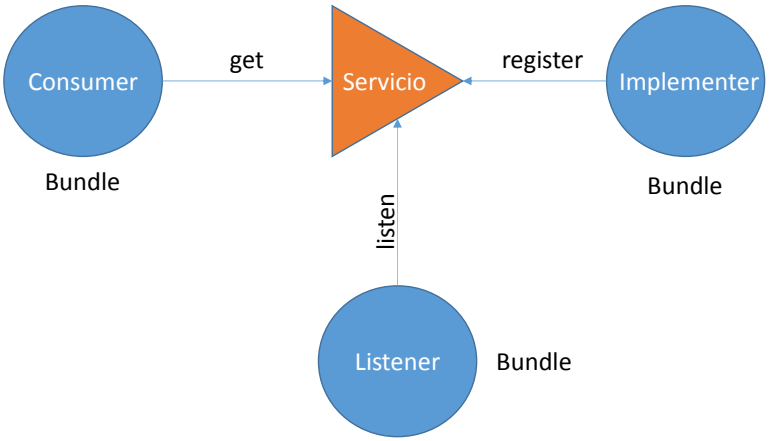

Figura 8. Operaciones de registro de servicios.

- ***Implementer*:** un *bundle implementer* será el que exporte el servicio, creando una interfaz que define al objeto Java a exportar, y ofreciendo la referencia mediante una operación *register* (registro).
- ***Consumer*:** un *bundle consumer* obtendrá dicha referencia al servicio mediante una operación *get* (obtención).
- ***Listener*:** los *bundles listener* están a la espera de los servicios que se registran, mediante una operación de *escucha* (*listen*).

Despliegue de bundles

Los *bundles* son desplegados en un entorno OSGi, con un entorno de ejecución determinado para el *bundle*. Dicho despliegue no consiste únicamente en la creación de contenedores para los *bundles*, como en el caso de aplicaciones Java para servidores. Se define el despliegue como un entorno colaborativo, en el cual los *bundles* que corren sobre la misma máquina virtual pueden compartir código. Aparte, la gestión y el mantenimiento de los *bundle* se encuentra estandarizado.

2.3.2. Implementación

Si bien la implementación de los *bundles* depende del desarrollador, existen varias tecnologías que permiten crear contenedores OSGi para facilitar el desarrollo de código, y abstraer de ciertas operaciones al desarrollador.

Spring DM

Spring DM (*Dynamic Modules*, módulos dinámicos) permite la creación de módulos Java mediante contenedores ligeros para OSGi, usando para ello características como inyección de dependencias y AOP (*Aspect-Oriented Programming*, programación orientada a aspectos). Para la creación de esos contenedores se utiliza un fichero con estructura XML como el siguiente [16]:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd">
<!--  <bean/> definitions here  -->
</beans>
```

Blueprint

Blueprint utiliza los mismos conceptos que Spring, pero la estructura del XML difiere en algunos aspectos:

- La raíz del XML es <blueprint> y no <beans>
- Se define un espacio de nombres dado por XSD (*XML Schema Definition*) propios de Blueprint.

El aspecto de un fichero de este tipo sería el siguiente [17]:

```
<?xml version= "1.0" encoding="UTF-8"?>
<Blueprint xmlns="http://www.osgi.org/xmlns/Blueprint/v1.0.0">
   <bean id="accountOne" class="org.apache.aries.simple.Account" />
</Blueprint>
```

2.3.3. Principales ventajas de OSGi

OSGi, como se ha definido anteriormente, reduce la complejidad de estructuración de los sistemas software, ya que provee un entorno modular. En dicho entorno modular, como se ha definido previamente, los módulos tienen independencia unos de otros, lo que facilita la labor del desarrollador para hacer un escalamiento de los sistemas.

Las principales ventajas de OSGi están adecuadas a las características de las arquitecturas orientadas a servicios. Las ventajas más importantes [18] de esta tecnología son:

- Reducen la complejidad en el desarrollo de las aplicaciones, ya que permite crear componentes independientes que se puedan relacionar entre sí. Por lo tanto, es una tecnología que se adapta a los cambios del sistema.

- Proporciona una transparencia de acceso a los componentes de un sistema creado mediante OSGi. Además, dispone de herramientas para el despliegue de aplicaciones de una manera sencilla y rápida.

- Es interoperable, ya que trabaja sobre la máquina virtual de Java. Además, permite la incorporación de numerosos protocolos de comunicaciones.

2.3.4. Releases

La primera especificación de OSGi fue lanzada en el año 2000, y desde entonces la OSGi Alliance ha ido generando nuevas versiones de la especificación para adecuarse a los cambios y la inclusión de las nuevas tecnologías. En la Tabla 1 se muestran todas las versiones lanzadas y el tiempo en el que se lanzaron, según lo indicado en la OSGi Alliance [19].

Especificación	Fecha
OSGi Service Gateway Release 1	Mayo de 2000
OSGi Service Gateway Release 2	Octubre de 2001
OSGi Service Platform Release 3	Marzo de 2003
OSGi Service Platform Release 4	• Versión 4.1: Mayo de 2007 • Versión 4.2 (Core and Compendium versión): Septiembre de 2009 • Versión 4.2 (Empresarial): Marzo 2010 • Versión 4.3 (Conpendium and Residential versions): Mayo de 2012
OSGi Release 5	Junio de 2012
OSGi Release 6	Junio de 2014

Tabla 1. Tabla de releases de OSGi lanzadas hasta la fecha.

2.3.5. Implementaciones más comunes de la especificación OSGi

Entre las implementaciones de un motor basado en OSGi, la más utilizada es la creada por Apache, que recibe el nombre de Apache Felix. Dicha implementación se explica en el apartado 3 de esta memoria, al definir los componentes del ESB Apache ServiceMix.

2.4. MOM (*Message Oriented Middleware*)

Un software de intermediación orientado a mensajes (MOM, *Message Oriented Middleware*) se trata de cualquier infraestructura software que permita el envío y la recepción de mensajes en sistemas distribuidos [20]. De esta forma, las aplicaciones distribuidas con capaces de comunicar sus componentes, así como comunicarse con otras aplicaciones distribuidas.

Las características básicas de un middleware orientado mensajería [20] son:

- Arquitectura centralizada: al utilizar un *Broker* o servidor de mensajería para el envío y recepción de mensajes, se centraliza la arquitectura de este tipo de middleware. Esto se muestra en la Figura 9 en la cual podemos observar múltiples aplicaciones que implementan un middleware orientado a mensajes.

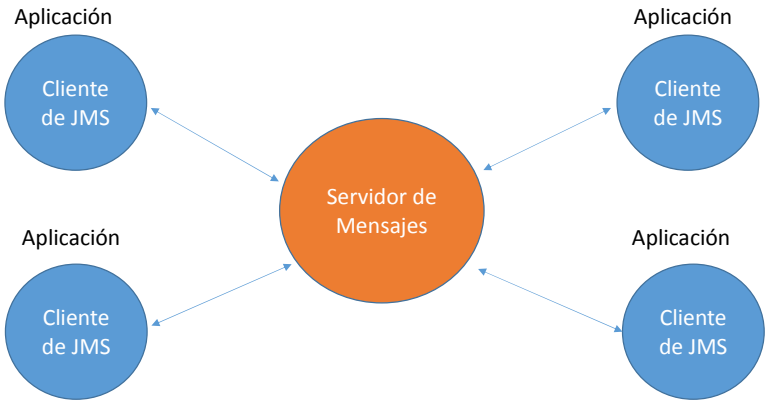

Figura 9. Arquitectura centralizada en MOM.

- Diseñado para comunicaciones asíncronas: el intercambio de mensajes tiene como finalidad el realizar aplicaciones que realicen operaciones no bloqueantes, de tal forma que las comunicaciones son totalmente asíncronas, que aumentan el rendimiento de las aplicaciones.
- Enrutamiento de mensajes: la mayoría de protocolos e implementaciones de este tipo de software de intermediación basan su funcionamiento en un sistema de colas de mensajes, sobre los cuales se puedan dirigir los mensajes a los destinatarios adecuados.
- Transformación de mensajes: a veces los mensajes deben ser transformados para poder adecuarse a los formatos utilizados por el emisor o el receptor. Este tipo de arquitectura provee herramientas para realizar esto.
- Permite a las aplicaciones tener un bajo acoplamiento, ya que no necesitan implementar un protocolo de comunicaciones.

Un middleware orientado a mensajes dispone de un elemento intermedio, llamado *Broker*, que es el que realiza las acciones principales del middleware. Dichas acciones se enumeran a continuación [21].

- Garantía de recepción, asegurando que todos los mensajes enviados llegan a su destino.
- Rapidez en la entrega de los mensajes.
- Aislamiento de la dificultad inherente de las comunicaciones entre aplicaciones, ya que el sistema se basa en mensajes, y no en protocolos con paquetes de datos que requieren cumplir unos estándares.
- Comunicación no bloqueante entre aplicaciones gracias al carácter asíncrono de las comunicaciones.
- Encolamiento de mensajes, proveyendo métodos para entregar los mensajes almacenados según el receptor esté disponible o no.
- Administración de las colas de mensajes, estableciendo criterios de ordenación y envío, como FIFO (*First Input First Output*, primero en entrar, primero en salir), o colas con prioridades.

En la Figura 10 se puede observar el funcionamiento básico de un *Broker*, que tiene una funcionalidad parecida a la que pudiera tener un *router*, pero siendo en este caso enrutamiento de mensajes.

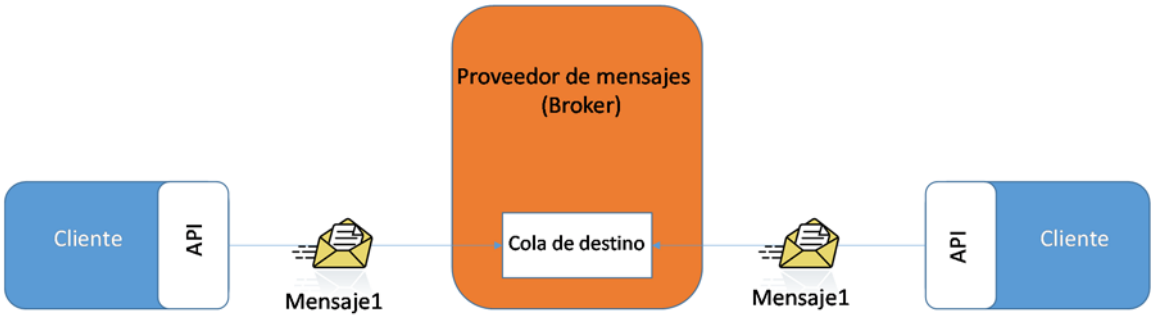

Figura 10. Función de un Broker de mensajería en un MOM.

El funcionamiento general de una comunicación sería el siguiente:

1. Un cliente envía un mensaje a un destino. Este destino es una cola.

2. El *Broker* encola el mensaje en la cola de destino.

3. Si en el momento de recepción del mensaje el destinatario está activo, cogerá de forma inmediata el mensaje de la cola. Si no se encuentra disponible, el mensaje se encola hasta que el receptor esté disponible.

4. El receptor recibe el mensaje.

Existen dos modelos de funcionamiento de este tipo de middleware. Estos dos modelos [22] son:

- Modelo punto a punto (PTP, *Point to Point*): en un modelo punto a punto, los mensajes enviados por el emisor son recibidos por un único receptor. En terminología MOM, es el productor el que genera los mensajes, los cuales serán recibidos por el consumidor, que toma los mensajes de la cola destino. Esta estructura se muestra en la Figura 11.

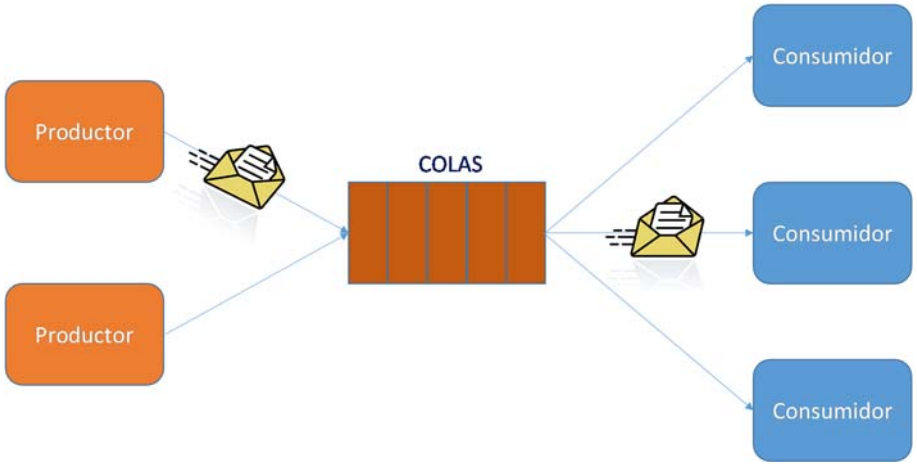

Figura 11. Modelo PTP en MOM.

- Modelo de publicador/subscriptor (*Pub/Sub*, *publisher/subscriber*): en un modelo de este tipo, un emisor envía un mensaje que será recibido por múltiples receptores. Utilizando la nomenclatura MOM, el emisor sería el publicador, y cada uno de los receptores serían suscriptores. La cola de destino de los mensajes se convierte en un tópico o asunto sobre el que los suscriptores obtienen los mensajes. Este modelo se representa en la Figura 12.

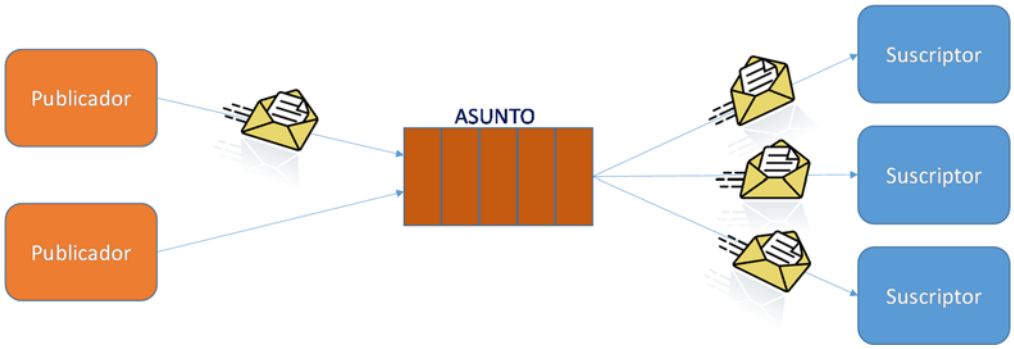

Figura 12. Modelo Pub/Sub de MOM.

La implementación más habitual y utilizada de un MOM es la solución JMS (*Java Message Service*, o Servicio de Mensajería de Java) diseñada por Sun Microsystems, que muestra una API que permite construir aplicaciones que utilicen un sistema de mensajería. La mayoría de implementaciones de middleware orientado a mensajes siguen las directrices mencionadas en la API de JMS.

Así, encontramos por ejemplo:

- MQSeries, diseñado por IBM.
- SonicMQ, diseñado por la empresa de software Progress.
- ActiveMQ, desarrollado por Apache.
- FioranoMQ, WebLogic Server, OpenJMS, y otras implementaciones menos utilizadas.

2.4.1. JMS (*Java Message Service*)

La API Java JMS establece interfaces y clases para la implementación de sistemas basados en mensajería, y que cumple con las especificaciones propias de un middleware orientado a mensajes. Existen numerosas implementaciones, que desarrollan sus propios proveedores de servicio. En la Figura 13 se muestran algunas de estas implementaciones, como Apache ActiveMQ o RabbitMQ.

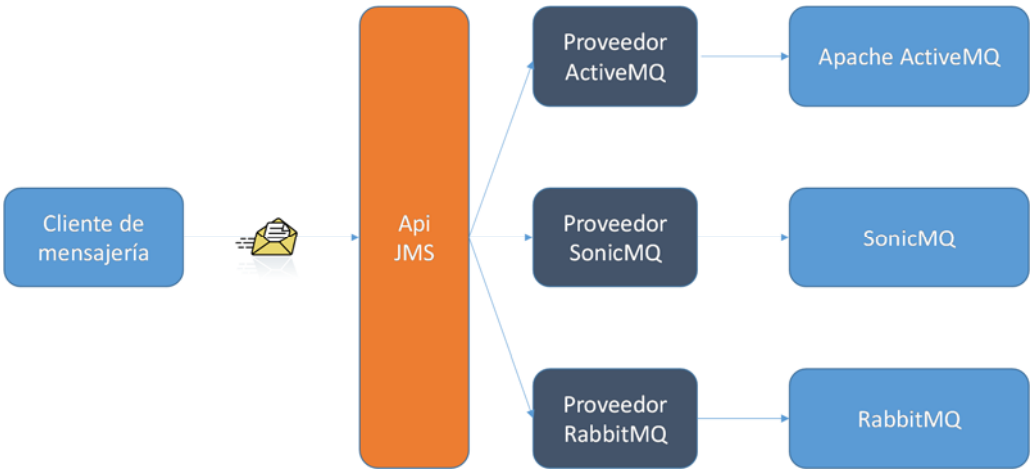

Figura 13. Implementaciones de JMS para desplegar un MOM.

De esta forma, la API JMS permite dar interoperabilidad entre las diferentes implementaciones de proveedores de sistemas de mensajería, ofreciendo unas interfaces comunes para la lógica de las comunicaciones.

2.4.1.1. *Arquitectura de JMS*

Para poder ofrecer una API ampliamente estandarizada, JMS define ciertos elementos que permiten englobar el concepto de mensajería. Estos elementos son:

- *Cliente JMS*: el cliente se trata de una aplicación, basada en Java, que envía y recibe mensajes. Un ejemplo sería cualquier componente de Java EE. Existen varios tipos:
    - *Productor JMS*: una aplicación cliente que crea y envía mensajes JMS.
    - *Consumidor JMS*: una aplicación cliente que recibe y procesa mensajes JMS.
- *Proveedor JMS*: implementación de los interfaces JMS el cual está escrito en su gran mayoría en lenguaje Java. El proveedor debe ofrecer prestaciones tanto de administración como de control de los recursos JMS. Toda implementación de la plataforma Java incluye un proveedor JMS.
- *Mensaje JMS*: elemento principal de JMS; objeto (cabecera + propiedades + cuerpo) que contiene la información y que es enviado y recibido por clientes JMS.
- *Dominio JMS*: los dos estilos de mensajería: PTP y *Pub/Sub*.
- *Objetos Administrados*: objetos JMS preconfigurados que contienen datos de configuración específicos del proveedor, los cuales utilizarán los clientes. Los clientes acceden a estos objetos mediante JNDI (*Java Naming and Directory Interface*, interfaz Java de directorio y nombrado):
    - *Factoría de Conexión*: los clientes utilizan una factoría para crear conexiones al proveedor JMS.
    - *Destino*: objeto (cola/asunto) al cual se direccionan y envían los mensajes, y desde donde se reciben los mensajes.

Estos elementos interaccionan de la manera expuesta en la Figura 14.

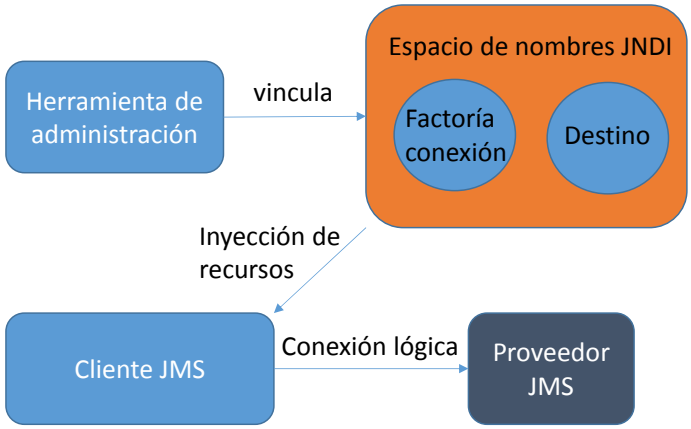

Figura 14. Interacción de elementos de JMS.

Las herramientas de administración permiten vincular destinos y factorías de conexión a través de un espacio de nombres JNDI. Entonces un cliente JMS puede consultar los objetos administrados en dicho espacio vía inyección de recursos y establecer conexiones lógicas con ellos a través del proveedor JMS.

2.4.1.2. *Estructura de los mensajes JMS*

Los mensajes JMS siguen un estándar específico. Como se muestra en la Figura 15 un mensaje está formado por tres partes principales, que permiten que el mensaje pueda ser dirigido a su destino llevando consigo los datos de usuario.

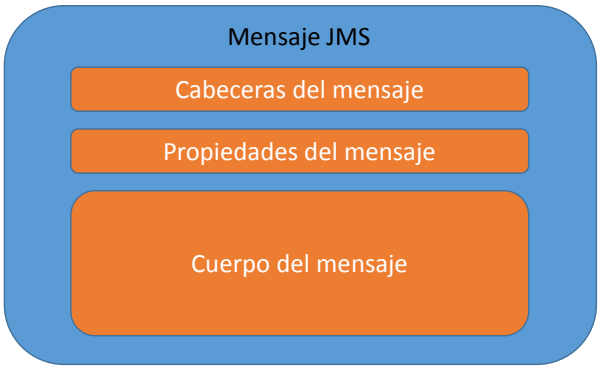

Figura 15. Estructura de un mensaje de JMS.

Los elementos que componen el mensaje [23] JMS son:

- **Cabecera**: utilizada por clientes y proveedores para poder identificar los mensajes.
- **Propiedades**: son propiedades en general para personalizar y/o hacer más específico un mensaje.
- **Cuerpo**: ss el mensaje en sí mismo, hay varios tipos de *cuerpos* que puede llevar un mensaje:
  - *StreamMessage*: Contiene un flujo (*stream*) de datos que se escriben y leen de manera secuencial.
  - *MapMessage*: Contiene pares nombre-valor.
  - *TextMessage*: Contiene un *String*.
  - *ObjectMessage*: Contiene un objeto que implemente la interfaz *Serializable*.
  - *BytesMessage*: Contiene un *stream* de bytes.

2.4.2. AMQP (*Advanced Message Queuing Protocol*)

El estándar AMQP (*Advanced Message Queuing Protocol* o Protocolo de Encolamiento de Mensajes Avanzado) define un protocolo de estándar abierto, que funciona en la capa de aplicación del modelo OSI (*Open System Interconnection* o Sistemas de interconexión abiertos).

Aunque JMS es una API robusta y madura, con una gran cantidad de implementaciones con soluciones muy sólidas, posee sólo de interoperabilidad en aplicaciones escritas en lenguaje Java [24]. Por tanto, las soluciones para aplicaciones que no estén escritas en Java normalmente requieren de sistemas propietarios de mensajería. Otra solución habitual es incluir transformaciones de los mensajes en los *Brokers* de mensajería, de tal forma que se adapten a los diferentes protocolos.

El problema de esta solución es que algunos mensajes contienen ciertos atributos que no pueden ser mapeados, y por lo tanto se puede perder información, e incluso en algunos casos hacer imposible la transmisión del mensaje.

Este problema se soluciona con la utilización de AMQP, que define un protocolo de mensajería que permite a aplicaciones heterogéneas interactuar entre sí. En la se muestran los dos escenarios en los cuales se puede mover AMQP: tanto en clientes basados en Java, como clientes basados en diferentes lenguajes.

Podemos observar los diferentes escenarios en los que puede trabajar la tecnología de AMQP en la Figura 16.

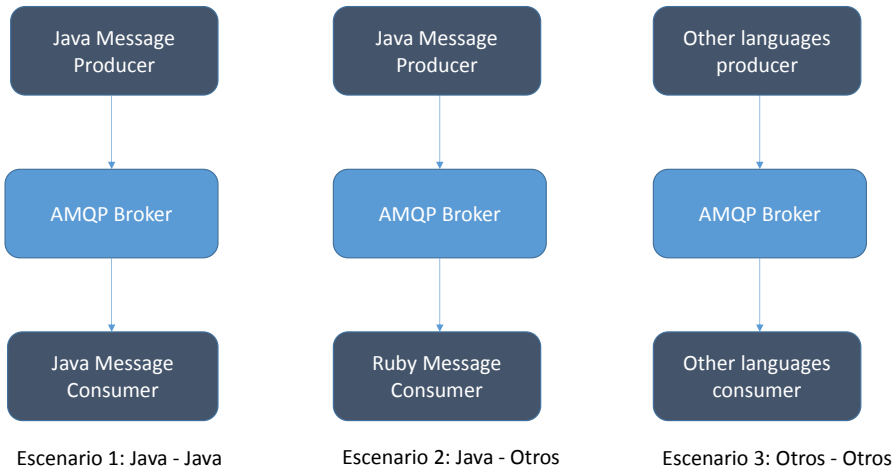

Figura 16. Posibles escenarios en AMQP.

2.4.2.1. *Características*

AMQP es un protocolo definido a *nivel de cableado*, que define el formato de los datos que se envían a través de la red como un flujo de octetos. De esta manera, cualquier aplicación que pueda crear e interpretar mensajes conforme a este formato de datos puede implementar el protocolo definido en AMQP. Se pueden citar como características de AMQP las siguientes [25]:

- **Eficiencia**: es un protocolo orientado a la conexión que utiliza una codificación binaria para las instrucciones del protocolo y los mensajes que se trasfieren a través de éste. Incorpora esquemas de control de flujo para maximizar la utilización de la red y de los componentes conectados. El protocolo está diseñado para encontrar un equilibrio entre eficacia, flexibilidad e interoperabilidad.
- **Fiabilidad**: permite el intercambio de mensajes con una variedad de garantías, desde la entrega tipo *envío y olvidar* hasta la entrega fiable, pasando por la entrega confirmada exactamente una vez.
- **Flexibilidad**: es un protocolo flexible que se puede usar para admitir distintas topologías. El mismo protocolo se puede utilizar para las comunicaciones cliente a cliente, cliente a agente y agente a agente.
- **Independiente del modelo de agente**: la especificación AMQP 1.0 no exige demasiados requisitos del modelo de mensajería que usa un agente. Esto significa que es posible agregar fácilmente compatibilidad con AMQP a los agentes de mensajería existentes.

2.4.2.2. *Componentes definidos en el estándar*

El estándar AMQP define una serie de entidades, que son las que interactúan entre sí para aplicar un sistema de mensajería. Desde el punto de vista de interconexión, se pueden definir estas entidades [25]:

- **El *Broker* de mensajería**: Comúnmente es llamado *Service bus*, o bus de servicios. Se trata de un servidor al que los clientes AMQP se conectan usando el protocolo AMQP. Los *Brokers* pueden ejecutarse en un entorno distribuido, pero esta capacidad es específica de la implementación y no está cubierta por la especificación. Se puede ver un ejemplo del uso del *Broker* en la Figura 17.

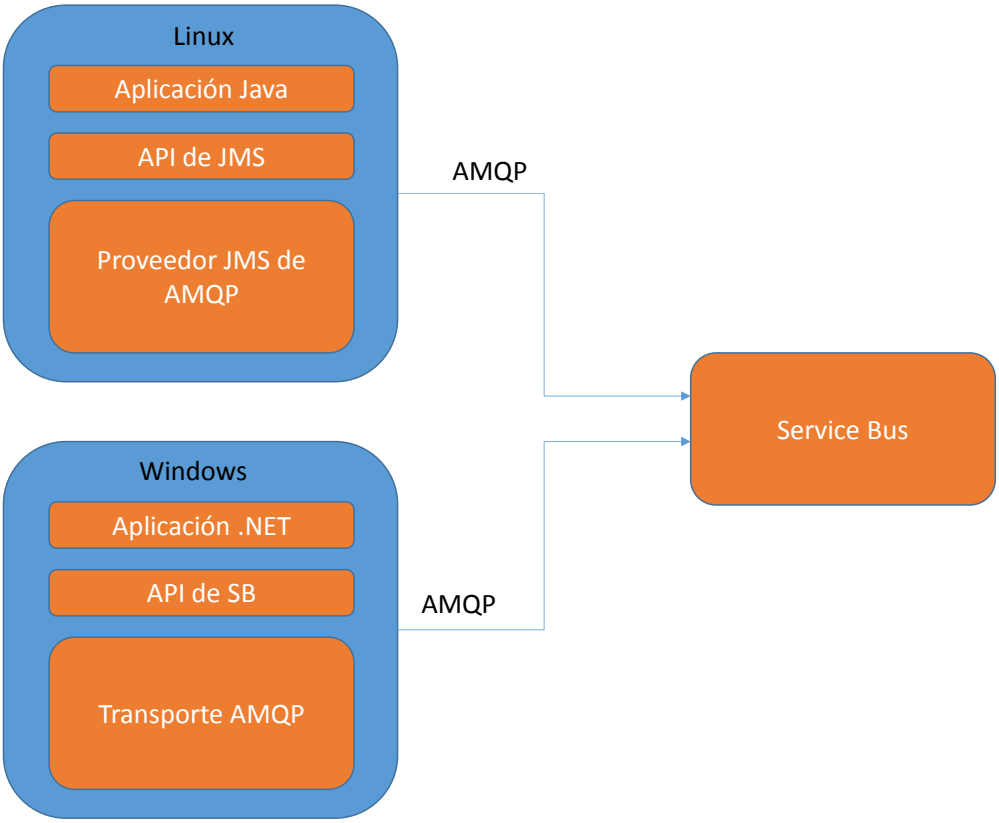

Figura 17. Service bus o Broker de AMQP.

- **Usuario**: un usuario es una entidad que, mediante la presentación de credenciales tales como una contraseña, puede ser autorizado (o puede no ser autorizado) a conectarse a un *Broker*.
- **Conexión**: una conexión física usando por ejemplo el protocolo TCP (*Transmission Control Protocol/Internet Protocol*, o Protocolo de Control de la Transmisión/Protocolo de Internet) o SCTP (*Stream Control Transmission Protocol*, o Protocolo de Control del Flujo de Transmisión). Una conexión está ligada a un usuario.
- **Canal**: una conexión lógica que está unida a una conexión física. Así pues, la comunicación a través de un canal posee un estado.

2.4.2.3. *Implementaciones de AMQP*

RabbitMQ

RabbitMQ es un software de mensajería de código abierto que implementa AMQP [26], que permite su utilización en sistemas distribuidos. RabbitMQ está formado por un servidor de intercambio (*Service bus*), las pasarelas para protocolos como HTTP (*Hypertext Transfer Protocol*, o Protocolo de Transferencia de Hipertexto) y STOMP (*Stream Text Oriented Messaging Protocol*, o Protocolo de Mensajería Orientado a Flujo de Texto), Librerías para Java y .NET, y diferentes *plugin*s que amplían sus funcionalidades.

ActiveMQ

ActiveMQ implementa un sistema de mensajería de código abierto, y fue creado por la fundación de software Apache [27]. A partir de la versión 5.8.0 de ActiveMQ, se implementa el estándar AMQP 1.0.

2.5. Servicios Web (*Web Service*s)

El término servicio web está referido a un conjunto de tecnologías que permite estandarizar aplicaciones que se mueven en el entorno web. Estas tecnologías interactúan entre ellas para poder ofrecer un servicio de la siguiente manera [28]:

- Un servicio se define mediante un WSDL, que tiene una estructura XML.
- Dicha definición del servicio se expone mediante una interfaz pública utilizando UDDI.
- El intercambio de información entre los servicios se hace mediante el protocolo SOAP.
- Si se quiere utilizar otra forma más directa de exponer los servicios, se pude utilizar una interfaz REST.

Las mencionadas tecnologías se explican a continuación.

2.5.1. XML

XML es un lenguaje definido por marcas, que permite desarrollar estándares para el intercambio de información estructurada entre diferentes plataformas.

Los datos que generan e intercambian los diferentes servicios web tienen una estructura definida por un lenguaje XML.

2.5.2. SOAP

SOAP es un protocolo de mensajería con una estructura de mensajes basada en XML. Mediante SOAP se puede transmitir información compleja mediante mensajes. Las características principales de este protocolo son:

- Es un protocolo que puede funcionar sobre cualquier protocolo de transporte, como HTTP, SMTP o incluso JMS.
- Es independiente del lenguaje de programación.

La estructura de un mensaje XML de SOAP es la siguiente:

```
<?xml version=”1.0” ?>
<env:Envelope xmlns=http://www.w3.org/2003/05/soap-envelope>
   <env:Header>
      <m:headers>…</m:headers>
   </env:Header>
   <env:Body>
      <m:element2>…</m:element2>
   </env:Body>
</env:Envelope>
```

En todos los mensajes de SOAP está presente el elemento raíz <Envelope>, en el cual se indica el espacio de nombres del documento. A continuación hay una serie de elementos de cabecera contenidos en el elemento <Headers>, en la cual se envían parámetros relativos para el procesado del mensaje.

El cuerpo propio del mensaje está contenido en el elemento <Body>, en el cual están contenidos los datos que se quieren enviar.

2.5.3. WSDL

WSDL es un lenguaje especificado en XML que permite definir a un servicio web. Define la interfaz pública del servicio para los usuarios. La definición de un servicio se realiza mediante un XML, que contiene los siguientes elementos estructurales:

- <types>, donde se definen los tipos de datos gestionados por el servicio que se describe.
- <message>, aquí se definen los elementos de los mensajes intercambiados en el servicio.
- <portType>, donde se definen las operaciones permitidas y los tipos de mensajes intercambiados a través del servicio que se describe.
- <binding>, aquí se especifican los protocolos utilizados para las comunicaciones, como SOAP.
- <service>, donde se establecen los puertos del servicio, y su dirección.

Utilizando WSDL en conjunto con UDDI (*Universal Description, Discovery and Integration.*) para la publicación de los servicios en la web, se puede acceder a un servicio a través de cualquier parte.

2.5.4. REST

Si bien el término REST (*Representational State Transfer)* se refería originalmente a un conjunto de principios de arquitectura, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP. Es posible diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST y también es posible diseñar interfaces XML/HTTP de acuerdo con el estilo de llamada a procedimiento remoto (RPC, *Remote Procedure Call*), pero sin usar SOAP.

Los sistemas que siguen los principios REST se llaman con frecuencia *RESTful*. Se puede observar en la Figura 18 la forma de interacción de diferentes servicios web *RESTful* [29].

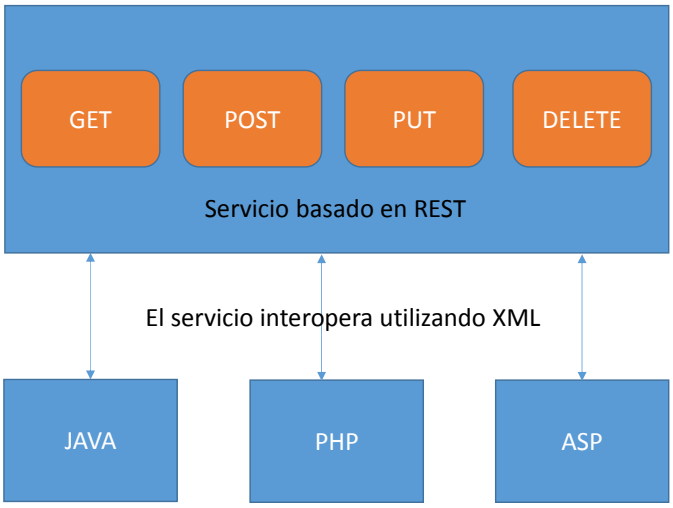

Figura 18. Web Service basados en interfaces REST.

REST ofrece una gran escalabilidad gracias a algunos conceptos como [30]:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs (*Uniform Resource Link*, o Enlace a Recurso Uniforme), no son permitidas por REST).
- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI (*Uniform Resource Identifier*, o Identificador de Recurso Uniforme).

2.6. Enterprise Service Bus

La forma más común de implementación de sistemas distribuidos mediante arquitecturas orientadas a servicios son las soluciones software de tipo ESB (*Enterprise Service Bus*, Bus de servicios empresariales).

Un ESB define un software de intermediación entre aplicaciones de sistemas distribuidos, mediante el uso de sistemas de mensajería, y el uso de un motor OSGi para el desarrollo de servicios. Un ESB debe cumplir las siguientes funcionalidades y requisitos [31].

- **Transparencia de ubicación**: se provee una abstracción del servicio, de tal forma que no se necesita saber su ubicación para acceder a él.
- **Conversión de protocolos**: un ESB debe de tener la capacidad de integrar de forma transparente a través de diferentes protocolos de transporte tales como HTTP(s), JMS, FTP (*File Transfer Protocol*, o Protocolo de Transferencia de Ficheros), SMTP, TCP, etc.
- **Transformación, enrutamiento y mejora de mensajes**: el ESB brinda funcionalidad para transformar mensajes desde un formato hasta otro formato basado en estándares tales como XSLT (*eXtensible Stylesheet Language Family Transformation*, o Transformación de Familia de Lenguajes de Hojas de estilo Extensible) y XPath. Además, debe ser capaz de enrutar mensajes hasta sus destinatarios, pudiendo añadir mejoras al mensaje.
- **Seguridad**: autenticación, autorización, y funcionalidad de encriptación se proveen a través del ESB para asegurar los mensajes entrantes. Igualmente estas funcionalidades se aplican a mensajes salientes para satisfacer requerimientos de seguridad del proveedor del servicio a consumir.
- **Monitoreo y Administración**: un ESB debe contener herramientas para la monitorización de los servicios y aplicaciones. Asimismo, debe disponer de funcionalidades extendidas para la administración de estos servicios y aplicaciones.

Se puede ver en la Figura 19 una perspectiva general de todas estas funcionalidades.

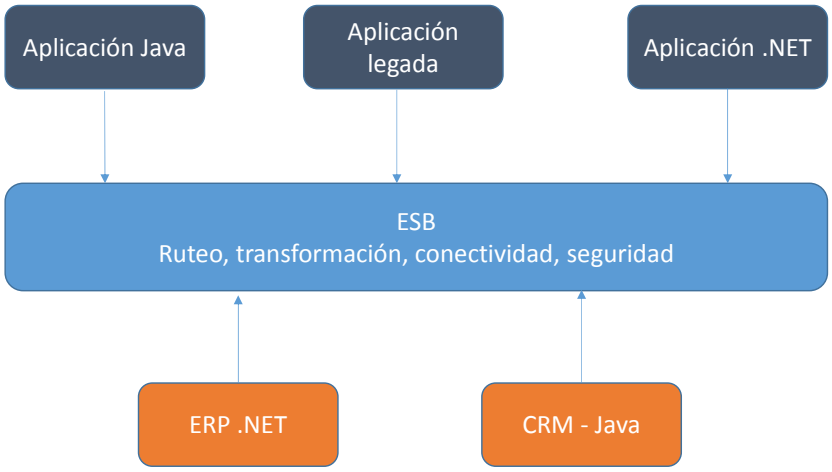

Figura 19. Perspectiva general de las funcionalidades de un ESB.

2.6.1. Componentes

Un ESB aúna componentes propios de una arquitectura orientada a servicios, entre ellos:

- Motor OSGi: el desarrollo de servicios y aplicaciones se hace mediante la implementación de las especificaciones definidas por la OSGi Alliance para realizar aplicaciones modulares. Se incluyen distintos tipos de contenedores OSGi, como Blueprint o Spring DM.
- MOM: se utiliza una implementación de alguna de las soluciones para middleware de mensajería, que permite la comunicación entre aplicaciones y sistemas.
- Funcionalidades añadidas: un ESB provee herramientas añadidas para poder utilizar los componentes propios de una arquitectura orientada a servicios, ofreciendo por ejemplo:
    - Consola de administración, que permite realizar operaciones sobre OSGi y otros componentes.
    - Sistema de log para registrar los eventos que puedan surgir en el sistema.
    - Herramientas para el despliegue de *bundles* y servicios.
    - Herramientas para monitorización y testeo de aplicaciones y servicios.
    - Componentes que proporcionan diferentes funcionalidades añadidas, como el incremento de seguridad.

2.6.2. Implementaciones

Existen numerosas implementaciones software para ESB. En este proyecto se realiza un estudio a fondo de las soluciones más comunes de ESB, en el capítulo 3 de este documento.

2.7. Middleware semántico

En entornos que son sensibles al contexto, los dispositivos que forman parte del sistema se comunican entre sí para el intercambio de información de estado, y otros parámetros que definen el contexto en el que se mueven. Un ejemplo de este tipo de sistemas son las *Smart Grid*, en las cuales los dispositivos que consumen energía son dispositivos ubicuos, y que cambian su estado de funcionamiento constantemente.

El uso de un middleware semántico permite crear una arquitectura para un sistema de forma que se proporcione una información detallada de contexto [32]. En soluciones basadas en servicios, este middleware administra servicios definidos a partir de una cierta ontología de servicios.

Debido a que es un concepto muy abstracto y de amplia definición, cada sistema dispondrá una forma diferente de integración de este tipo de middleware.

2.8. Ontología de servicios

Una ontología de servicios se puede definir como una especificación formal y explícita de una conceptualización compartida de diferentes tipos de servicio. Dichos servicios pueden tratarse como dominios de conocimiento [33]. En otras palabras, se trata de un conjunto de conceptos que representan un dominio de conocimiento.

Una ontología de servicios es:

- **Formal,** ya que permite la interactuación de máquinas.
- **Explícita,** ya que los conceptos, relaciones y axiomas son definidos de forma explícita.

- **Compartida,** que permite abstraer y simplificar un modelo de servicio mediante el consenso del conocimiento.

Los componentes básicos de un concepto son:

- Los términos, que son los nombres con los que se refieren cada uno de los conceptos de un dominio de conocimiento. Se representan mediante clases y subclases.
- Las propiedades, que definen y describen en detalle el concepto.
- Las relaciones, que proveen las interacciones entre los diferentes conceptos que forman un dominio de conocimiento.

Estos componentes de un concepto se muestran en la Figura 20.

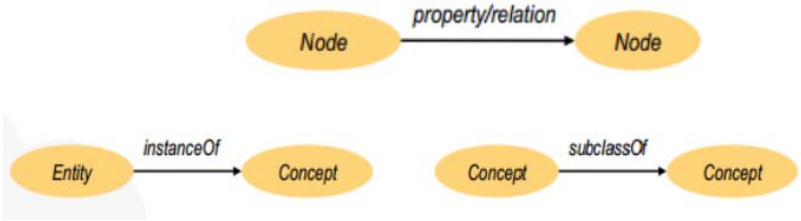

Figura 20. Componentes y su interacción en una ontología.[2]

Algunas de las ventajas derivadas del uso de ontologías de servicios son:

- La compartición de la estructura de la información entre usuarios o agentes software.
- Reutilización de los dominios de conocimiento (servicios).
- Dominios definidos de forma explícita.
- Separa el dominio de conocimiento (servicio) del operacional (lógica de negocio).
- Permite un análisis profundo del dominio de conocimiento.

[2] Fuente: *Semántica para las Smart Cities.* Asignatura *Aplicaciones Telemáticas Avanzadas.* ETSIST, 2014 [33].

3. Entorno de desarrollo del proyecto

El objetivo principal de este proyecto es evaluar la posibilidad de utilizar como soporte hardware para el controlador de dispositivos al ordenador *Raspberry Pi*, el cual es un mini ordenador de dimensiones muy reducidas, que tiene un gasto de energía ínfimo comparado con el de un ordenador personal. En los siguientes apartados se muestra una definición más amplia y detallada de este dispositivo.

En la evaluación del *Raspberry Pi* se va a instalar y configurar un *Enterprise Service Bus* para, a continuación, desarrollar y probar un controlador de sensores y actuadores acorde a la definición de la arquitectura *e-GOTHAM*.

En los siguientes apartados se describen el *Raspberry Pi* y el entorno de desarrollo *e-GOTHAM*.

3.1. *Raspberry Pi*

Raspberry Pi es el nombre que recibe un modelo de ordenador de placa reducida (SBC, *Single Board Computer*) lanzado en 2011. El proyecto *Raspberry Pi* surge en 2006 de la mano de Eben Upton, Rob Mullins, Jack Lang y Alan Mycroft, cuando deciden crear un ordenador de bajo coste orientado a niños para favorecer la enseñanza de conocimientos informáticos entre los alumnos más jóvenes. Así, en 2009 crean la organización caritativa *Raspberry Pi Foundation.*

En agosto 2011, un primer modelo *Alpha* del *Raspberry Pi* es lanzado, mostrando algunas de las características que posteriormente serían las que integrarían el prototipo final.

En febrero de 2012 empieza la comercialización del *Raspberry Pi* en dos modelos, A y B, los cuales se especificarán en el siguiente apartado.

3.1.1. Especificaciones

En la Figura 21 se puede ver el aspecto real de un *Raspberry Pi* modelo B.

Figura 21. Placa Raspberry Pi modelo B.

El *Raspberry Pi* está construido como una SBC con una comercialización en dos modelos: el modelo A y el modelo B.

El modelo A posee 256 MiB de memoria RAM (*Random Access Memory*, o Memoria de Accceso Aleatorio), mientras que el modelo B posee 512 MiB. Por otro lado, El modelo B dispone de un puerto USB (*Universal Serial Bus*, o Bus Serie Universal) adicional mediante el uso de un HUB integrado, y de un puerto Ethernet 10/100 Mbit que utiliza este mismo HUB. Estas pequeñas diferencias hacen que el modelo A tenga un PVP (Precio de Venta al Público) y un consumo más reducido que el modelo B, que en cambio es más potente.

Cada una de las placas dispone de un *System On A Chip* fabricado por Broadcom [34], que incluye en un mismo chip el procesador central o CPU, la memoria RAM principal del sistema, y la unidad de procesamiento de gráficos o GPU (*Graphics Processing Unit*), junto con otros componentes como un procesador digital de señales (DSP, *Digital Signal Processor*) y los controladores de los puertos I/O (*Input/Ouput*, o Entrada/Salida).

Hay que mencionar la placa no posee un reloj en tiempo real, como el que podríamos encontrar presente en un PC, ya que no existe una alimentación permanente mediante algún dispositivo como puede ser la pila de botón existente en los PC. Sin embargo, es posible añadir un reloj externo mediante la inclusión de un dispositivo externo, como el DS1307 u obteniendo la hora a través de un servidor de hora en red.

El modelo A posee un modelo de este chip con un solo USB. En el modelo B, este único puerto USB es sustituido por dos conexiones USB más un conector RJ45, que funcionan a través de un HUB integrado en la placa para los 3 conectores.

Cabe destacar que este chip posee una GPU con una aceleración por Hardware que es capaz de codificar vídeo H.264 en 1080p.

El almacenamiento para la instalación del sistema operativo y los datos de usuario se consigue mediante el uso de tarjetas SD/MMC (*Secure Digital/Multimedia Card)*.

En la Tabla 2 podemos observar las características técnicas del hardware de cada uno de los modelos de comercialización.

Característica	Modelo A	Modelo B
SoC	BCM2835	
CPU	• ARM_(*Advanced RISC Machines*) 1176JZF-S a 700 MHz (familia ARM11), ARM v6 • Caché L2 128 KiB (*Kibibyte)* • Juego de instrucciones RISC (*Reduced Intruction Set Computer*, u Ordenador con Juego de Instrucciones Reducido) de 32 bit	
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG[3]-2 y VC-1, 1080p30 H.264/MPEG-4 AVC	
Memoria SDRAM	256 MiB compartida con GPU	512 MiB compartida con GPU
Almacenamiento	Almacenamiento mediante tarjetas SD	
Conexiones	1xUSB HDMI (*High Definition Multimedia Interface*, o Interfaz Multimedia de Alta Definición. RCA (PAL/NTSC[4]) Alimentación MicroUSB GPIO	2xUSB (Vía HUB integrado) RJ45(*Registered Jack 45*) (Vía HUB USB) HDMI RCA (PAL/NTSC) Alimentación MicroUSB GPIO
Consumo energético	2,5 W, 500 mA, 5V	3,5 W, 750 mA, 5 V
Dimensiones	85.60mm × 53.98mm	

Tabla 2. Especificaciones del Raspberry Pi.

[3] MPEG (Moving Pictures Experts Group)

[4] PAL/NTSC son las diferentes formas de visualización de contenidos de vídeo para pantallas analógicas.

En la Figura 22 se puede ver un resumen gráfico de los puertos y dimensiones del *Raspberry Pi* modelo B.

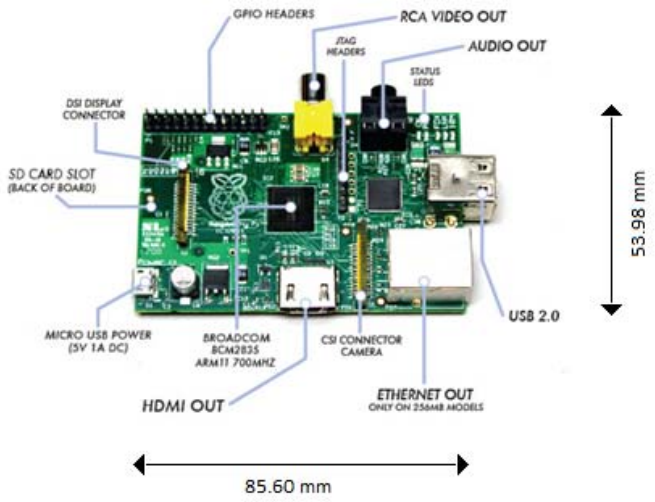

Figura 22. Conectores y dimensiones del Raspberry Pi.

3.1.2. Accesorios necesarios

Para poder interactuar y desarrollar el sistema necesario para el controlador central, necesitamos algunos dispositivos aparte del propio *Raspberry Pi*. Entre los dispositivos necesarios encontramos los definidos a continuación.

HUB USB de siete puertos

Debido a la poca energía proporcionada por los conectores USB del *Raspberry Pi*, es necesario utilizar un dispositivo con alimentación separada para poder conectar dispositivos adicionales a través de USB, ya que si no se pueden producir fallos de alimentación y alterar el funcionamiento normal del sistema.

Siguiendo recomendaciones de las páginas oficiales de *Raspberry Pi* [35], se ha elegido el modelo de HUB USB de siete puertos DUB-H7 de la marca D-Link, que se muestra en la Figura 23.

Figura 23. HUB de 7 puertos USB[5].

[5] Fuente: http://www.dlink.com/us/en/home-solutions/connect_us/usb/dub-h7-7-port-usb-2-0-hub

Teclado y ratón USB genéricos, y monitor con HDMI

Para poder interactuar con el *Raspberry Pi*, necesitamos un teclado y ratón con interfaz USB, ya que la interfaz PS/2 (cada vez menos extendida) no es soportada por el *Raspberry Pi* al no disponer de un puerto de este tipo.

Por otro lado, se necesita un monitor con un puerto HDMI para poder visualizar la información generada por el *Raspberry Pi*.

Tarjeta SDHC (Secure Digital High Capacity)

Como soporte de almacenamiento, el sistema utiliza tarjetas de tipo SD/MMC. Se ha decidido utilizar una tarjeta SDHC de clase 10, que son las de más rápido acceso de lectura y escritura de esta generación. En concreto, se ha utilizado el modelo SD10V de 16 GiB (*Gibibyte*) de memoria de la marca Kingston. Las características de esta tarjeta SD son las mostradas en la Figura 24.

Figura 24. Especificaciones de la tarjeta SD[6].

Accesorios opcionales

Aunque estos accesorios no son estrictamente necesarios, es interesante incluir algunos de estos accesorios para poder desarrollar funcionalidades interesantes ofrecidas por el *Raspberry Pi*:

- Adaptador inalámbrico para WiFi (*Wireless Fidelity)*: nuestro sistema administra dispositivos ubicuos, de tal forma que debe ser lo menos sensible a una dinamización de la ubicación de los dispositivos. La utilización de controladores centrales que dispongan de un método de comunicación inalámbrico permite una mayor libertad a la hora de su despliegue. De esta forma, la utilización de un adaptador inalámbrico para el acceso a redes WiFi provee al *Raspberry Pi* de una mayor libertad de ubicación. En el manual de usuario se detalla la forma de instalación de uno de estos adaptadores inalámbricos vía USB.

[6] Fuetne: http://www.kingston.com/datasheets/sd10v_es.pdf

- Dispositivos para puerto GPIO (*General Purpose Input/Output*, o Entrada/Salida de Propósito General): el puerto GPIO del *Raspberry Pi* permite la utilización de numerosos dispositivos digitales a partes de la integración de diferentes pines de comunicación digital, y puertos de alimentación que funcionan 5 V. De esta forma, podemos diseñar dispositivos como cámaras o paneles de visualización que añadan nuevas funcionalidades a nuestro sistema.

3.1.3. Soporte software

Desde su lanzamiento, existen numerosos Sistemas operativos disponibles para su instalación en el *Raspberry Pi*, mayoritariamente basados todos ellos en el núcleo Linux.

Algunos de los más destacados [35] son los siguientes:

- Arch Linux ARM.
- Pidora, versión de Fedora optimizada.
- Raspbian, versión de Debian Wheezy modificada para trabajar sobre procesadores ARMv6.
- FreeBSD, basado en Unix.

Entre todos estos sistemas operativos listados, se ha elegido instalar Raspbian como soporte operativo para el *Raspberry Pi* por las razones son las siguientes:

- Sistema basado en Linux, siendo una distribución modificada de Debian. Debian fue de las primeras distribuciones basadas en el núcleo Linux, por lo que está muy depurada, y existe una gran cantidad de paquetes de instalación, y soporte para una gran cantidad de dispositivos.
- Es el sistema más utilizado para operar sobre el *Raspberry Pi*, por lo que el soporte y la documentación existente es mucho mayor que para el resto de sistemas operativos disponibles.
- Es el único de los sistemas operativos que ha permitido instalar sobre él un ESB con ciertas garantías de funcionamiento.

Aparte del sistema operativo, se ha necesitado instalar ciertas aplicaciones que se requieren para implementar el ESB, y para el desarrollo de aplicaciones para éste:

- Se ha necesitado instalar un servidor de FTP. La razón es que es necesario obtener ciertos archivos de otros equipos, como puede ser los *bundles* desarrollados en dichos equipos que se ejecutarán en el ESB. Aunque estos *bundles* pueden ser desarrollados en el propio *Raspberry Pi*, al tratarse de un sistema con una potencia limitada, se hace pesado su desarrollo. Por ello, se ha optado por su desarrollo en un entorno integrado de desarrollo (como Eclipse) en un ordenador personal, para posteriormente pasar el archivo final mediante FTP al sistema *Raspberry Pi*.
- La instalación de la versión 1.6 de Java es un requisito indispensable de funcionamiento para el ESB, por lo que es necesario instalarla previamente. En el manual de usuario anexo se detalla la instalación del entorno JDK.

Por último, y como elemento más importante para el funcionamiento del sistema como controlador de dispositivos, se ha instalado un ESB. El ESB elegido es la versión 4.5.3 *minimal* de Apache ServiceMix, a la cual se añaden los módulos necesarios para ofrecer las funcionalidades requeridas.

La justificación de elección de este ESB viene explicada de forma detallada en el capítulo 4 de esta memoria, en la cual se exponen sus características frente a otros ESB disponibles.

3.2. El entorno de desarrollo *e-GOTHAM*

Una vez instalado y configurado el ESB, se va a implementar un controlador de dispositivos físicos. Este controlador de dispositivos permitirá gestionar diferentes sensores y actuadores, de los cuáles se podrán obtener medidas o, en general, interactuar de acuerdo a su funcionalidad. Como se ha mencionado en capítulos anteriores el entorno de desarrollo del controlador ha sido el proporcionado por la infraestructura del proyecto europeo de I+D *e-GOTHAM*.

Las funcionalidades específicas del controlador de dispositivos se enumeran en el capítulo 4 de este documento, concerniente al proceso de desarrollo software del controlador de dispositivos.

De una forma resumida, la Figura 25 ilustra los principales componentes de la arquitectura utilizada en el desarrollo del proyecto.

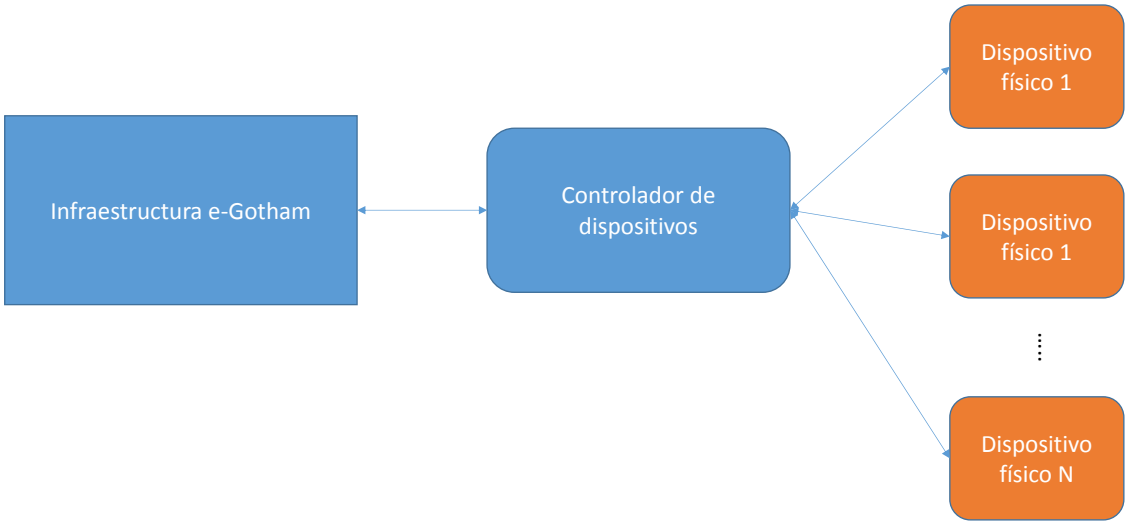

Figura 25. Estructura de componentes del proyecto.

En el bloque denominado Infraestructura *e-GOTHAM* existirá al menos un Controlador Central que soportará acciones de registro y toma de medidas de/desde el Controlador de Dispositivos.

3.2.1. Controlador de dispositivos

El controlador de dispositivos implementado tiene como objetivo la recogida de datos de diferentes dispositivos físicos ligados a él. Puede implementar otras funcionalidades como son:

- Alojar servicios o aplicaciones en general instaladas sobre el ESB.
- Registrar diferentes dispositivos físicos que se añadan al sistema, de tal forma que pueda comunicarse con ellos y administrarlos.
- Actuar como intermediario entre los dispositivos físicos y el usuario, permitiendo a éste interactuar con cada uno de los dispositivos conectados al controlador mediante una interfaz basada en aplicaciones.

3.3. Dispositivos físicos

Los dispositivos físicos son los dispositivos finales del sistema, sobre los cuales este actúa, realizando medidas y actuando sobre ellos.

Los dispositivos físicos abarcan una gran cantidad de posibilidades. Entre algunos de estos dispositivos, podemos encontrar:

- Sensores: diferentes sensores, que pueden medir atributos físicos como temperatura, corriente, voltaje, etc. También se puede actuar sobre redes de sensores inalámbricos, y otras muchas disposiciones de dichos sensores.
- Actuadores: permiten actuar sobre un sistema físico a partir de una orden recibida en forma de señal eléctrica.
- Transductores: convierten diferentes tipos de energía. Así, serán habituales transductores que conviertan la energía eléctrica en cualquier otro tipo de energía, como podría ser energía mecánica.

Aunque existen muchos más dispositivos de este tipo que no se contemplan, la naturaleza de los dispositivos físicos no es objeto de estudio en este proyecto.

En el siguiente apartado se describe el dispositivo físico utilizado para la realización de este PFG.

3.3.1. Soporte hardware

Como dispositivo físico, en este proyecto se ha utilizado el implementado en el PFC (Proyecto Fin de Carrera) sobre redes de sensores inalámbricos: *Gestión del conocimiento en redes de sensores inalámbricos* [36].

En el PFC mencionado se toman medidas de corriente (Figura 26) utilizando placas Arduino (Figura 27) integradas en una WSN (*Wireless Sensor Network*, o Red de Sensores Inalámbricos) que se conecta al *Raspberry Pi* mediante USB desde el dispositivo Arduino que actúa de sumidero (Figura 28).

Figura 26. Cabezal medidor de corriente en un cable.

Arduino [37] es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

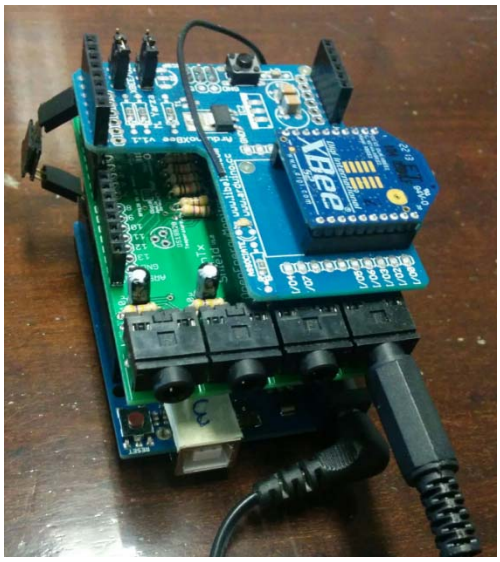

Figura 27. Arduino *con medidor de corriente.*

3.4. Entorno específico de desarrollo.

Una vez se ha visto la estructura general del sistema, la funcionalidad de todos los componentes, y la funcionalidad específica del componente desarrollado, en este apartado se procede a describir el entorno específico elegido para implementar este sistema.

Figura 28. Raspberry Pi y Arduino *sumidero.*

En la Figura 28 se muestra una foto del sistema general, donde se muestra el *Raspberry Pi* y el Arduino que actúa como sumidero de la WSN.

La Figura 29 esquematiza la interconexión de todos los elementos hardware utilizados en el desarrollo de este PFG.

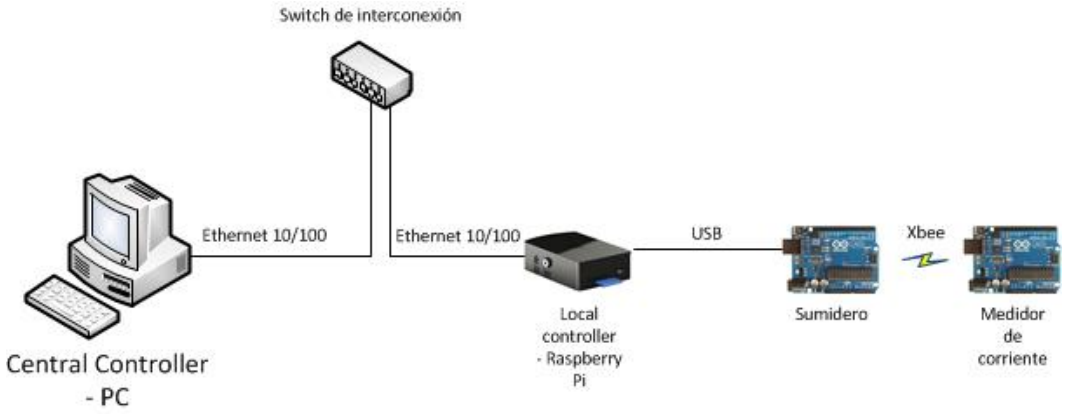

Figura 29. Diagrama de red del entorno específico.

Se pueden distinguir los siguientes elementos:

- *Central Controller-PC*. Elementos propios de la infraestructura de *e-GOTHAM*, para el desarrollo de las funcionalidades del controlador de dispositivos.
- *Raspberry Pi*. Como ya se ha definido en apartados anteriores, es el elemento elegido para comprobar su posible funcionamiento como controlador local.
- Red de Arduino. Las placas Arduino serán las encargadas de conformar el dispositivo físico. Encontramos dos dispositivos:
    - Uno de los Arduino será el sumidero de la red de sensores inalámbricos, el cual recibirá periódicamente las lecturas de cada uno de dichos sensores, y del cual el *Raspberry Pi* deberá ser capaz de obtener las lecturas.
    - El segundo Arduino posee un sistema para la lectura de datos de corriente alterna de un cable especialmente diseñado para este propósito.

4. Instalación de un ESB en un *Raspberry Pi*

Para poder desplegar la arquitectura orientada a servicios de nuestro sistema, se necesita el software necesario en cada uno de los componentes que conforman el sistema para poder implementar esta arquitectura.

Como se explica en el capítulo 2 de esta memoria, en este PFG la implementación software de las arquitecturas orientadas a servicios se realiza mediante programas denominados ESB (*Enterprise Service Bus*), que funcionan como intermediarios entre aplicaciones de diferentes sistemas, y se encargan de registrar y dar acceso a los servicios desarrollados.

La mayoría de estos ESB sólo requieren del sistema operativo y unas cuantas herramientas de software para funcionar, así como un entorno de ejecución basado en Java (en la mayoría de los casos). Sin embargo, requieren un hardware más o menos potente dependiendo de cada ESB. La mayoría requiere un mínimo de entre uno y dos GiB de RAM, y la mayoría establece un procesador de doble núcleo como requisito recomendado.

El *Raspberry Pi*, como se explica en el capítulo 3, posee unas especificaciones muy limitadas con respecto a los requisitos hardware mínimos que se mencionan. Por tanto, al ser el ESB un componente indispensable de la arquitectura, surge la duda de si se puede utilizar uno de estos programas software en este ordenador de dimensiones reducidas.

En este proyecto se busca estudiar la posibilidad de utilizar al *Raspberry Pi* como herramienta de despliegue, utilizándola como un controlador de dispositivos. Al ser un ESB el elemento imprescindible, el primer paso debe ser la elección de un ESB entre los disponibles actualmente.

En este apartado se llevarán a cabo la consecución de los siguientes objetivos:

- Estudio de ESB disponibles en la actualidad.
- Comparativa entre todos los ESB estudiados.
- Criterios de elección del ESB para el *Raspberry Pi*.
- Alternativa elegida: justificación de la elección.
- Descripción de las funcionalidades y componentes del ESB escogido.

Con el auge de las arquitecturas orientadas a servicios, sobre todo en entornos empresariales, han surgido numerosas soluciones que implementan dicha arquitectura. Cabe destacar que la mayoría de las soluciones actuales son de código abierto, con licencias software públicas. Sin embargo, también existen soluciones que no son de código abierto y con licencias propietarias, desarrolladas por empresas poderosas como IBM, Microsoft u Oracle, con un software muy pulido y con una gran cantidad de funcionalidades.

Aunque en este proyecto uno de los requisitos fundamentales será el uso de software de código abierto, se ha decidido hacer una pequeña aproximación también a las soluciones propietarias, que servirán para el estudio y comparación de los ESB elegidos.

En este apartado se definen las diferentes soluciones tanto de código abierto como de código no abierto, exponiendo sus características generales.

4.1. *Open source* y licencias

Las soluciones *open source* permiten al poseedor del producto su completa modificación, redistribución, copiado y, en definitiva, cualquier operación sobre el producto software. No hay que confundir *open source* con gratuito, ya que pueden tener un precio de venta al público.

Aunque se da una libertad casi completa en el uso del software en las soluciones de *open source*, existen licencias software que contemplan algunos usos y situaciones. En ellas se definen el marco de uso de un programa software. Las que se analizan en el documento son las siguientes:

- **Apache Software License**: se trata de una licencia de software libre creada por la ASF (*Apache Software Foundation*, Fundación de Software Apache) para los productos Apache y otros productos que utilicen software diseñado por Apache. Esta licencia garantiza sólo obliga a notificar el copyright de Apache en la distribución de su software.
- **GPL**: son las siglas de *General Public License* o Licencia Pública General. Es una licencia de software libre con *copyleft*[7].
- **AGPL (*Affero General Public License,* o Licencia Pública General Affero)**: se trata de una versión de la licencia GNU GPL que añade la obligación de distribuir software si este se ejecuta para ofrecer servicios a través de una red de ordenadores.
- **LGPL (*Lesser Development and Distribution License* o Licencia Menor De Distribución y Desarrollo)**: la principal diferencia con GPL es que estas licencias se pueden enlazar con software que utiliza licencias no-GPL.
- **CDDL (*Common Development and Distribution License,* o Licencia Común de Distribución y Desarrollo)**: esta es una licencia de software libre. Tiene *copyleft* con un alcance similar al de la Licencia Pública de Mozilla, lo que la hace incompatible con la GPL de GNU. Esto significa que un módulo cubierto por la GPL y otro cubierto por la CDDL no pueden ser legalmente enlazados entre sí.
- **CPAL (*Common Public Attribution License,* o Licencia Púlbica de Atribuciones Común)**: esta es una licencia de software libre. Se basa en la versión 1 de la Licencia Pública de Mozilla y es incompatible con la GPL por las mismas razones: posee varios requisitos para las versiones modificadas del software que no existen en la GPL. También exige que se publique la fuente del programa si se va a permitir su uso.

[7] El *copyleft* permite la distribución y modificación de una obra u otro trabajo, en el cual se garantiza los mismos derechos para las copias modificadas.

4.2. Estudio de los ESB disponibles.

En los siguientes apartados se definen los ESB de este tipo más utilizados en la actualidad, exponiendo sus características generales.

4.2.1. Apache ServiceMix

ServiceMix es el software SOA desarrollado la fundación de desarrollo de software llamada Apache. Se trata de un sistema ESB flexible y de código abierto, que aúna otros proyectos desarrollados por Apache para ofrecer las funcionalidades propias de un sistema orientado a servicios.

Las características principales son:

- Software de código abierto.
- Gratuito.
- Funciona sobre una JVM (*Java Virtual Machine*, o Máquina Virtual Java).
- Licencia Apache Software License.

Entre las funcionalidades que tiene, se puede encontrar:

- Sistema de mensajería fiable entre aplicaciones.
- Sistema de ruteo y generación de mensajes.
- Implementación de EIP (*Enterprise Integration Patterns*).
- Entorno OSGi.
- Despliegue de *bundles* en caliente.
- Sistema de log de mensajes.
- Cumple con las especificaciones JBI (*Java Business Integration*¸o Integración de Negocio en Java).
- Consola de administración web.

En la Figura 30 se muestra una estructura general de los componentes de ServiceMix.

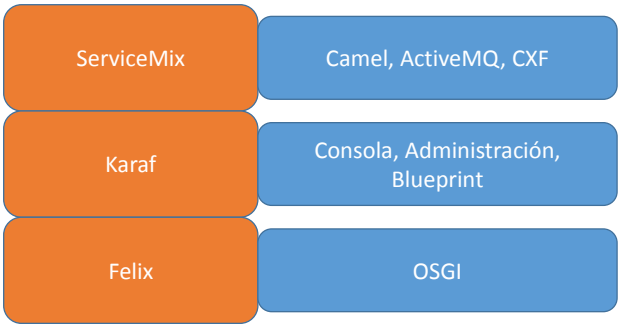

Figura 30. Estructura de ServiceMix.

La última versión del programa lanzada es la 5.0. Sin embargo, se utilizará la versión 4.5.3 para su estudio, ya que era la última versión lanzada en el momento de elección, y porque sigue siendo la versión más estable y con más actualizaciones y documentación disponibles.

Establece los siguientes requisitos mínimos:

- Java Development Kit 1.6.x.
- 100 MiB (*Mebibyte)* de espacio de disco libre.
- Se recomienda un mínimo de 1 GiB de memoria RAM.

Los datos expuestos sobre ServiceMix se corresponden con los presentes en la página oficial de Apache ServiceMix [38].

4.2.2. Fuse ESB Enterprise

FuseSource era una compañía de desarrollo de software que hasta 2012 desarrolló el ESB Fuse EB Enterprise para proveer una solución SOA para aplicaciones empresariales.

En 2012 la compañía empieza a formar parte de Red Hat, una empresa de desarrollo software bajo entornos Linux.

La última versión de este ESB lanzada fue la 7.1.0, la cual posee las siguientes características:

- Código abierto.
- Gratuito.
- Funciona sobre una JVM (Java Virtual Machine).
- Licencia Apache Software License.

Este ESB tiene como núcleo de funcionamiento el Apache ServiceMix, al que añade algunas nuevas funcionalidades [39]. Así, añade al ESB de Apache:

- Nueva interfaz de administración web.
- Mejor administración de errores.
- Mayor soporte y documentación.

En la Figura 31 se puede observar la estructura y los componentes de Fuse ESB.

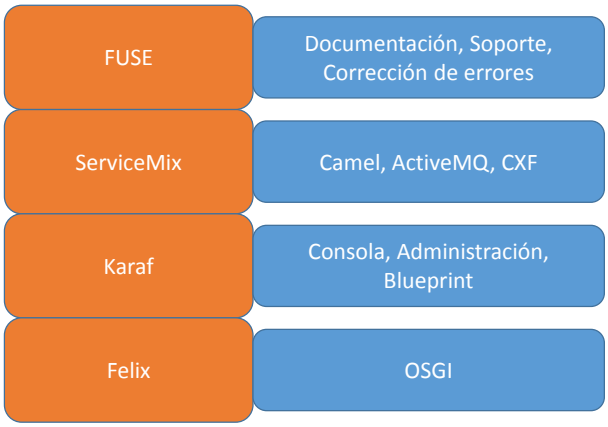

Figura 31. Estructura de Fue ESB.

Los requisitos mínimos y recomendados para este programa software son los mismos expuestos para ServiceMix, excepto que requiere un mínimo de 2 GiB de RAM para su correcto funcionamiento.

4.2.3. JBoss Fuse

En 2012 Red Hat compra la compañía fusesource, y desarrolla un ESB basado en el fuse ESB Enterprise diseñado por fusesource, que pasa a llamarse JBoss Fuse. La última versión de este programa se corresponde con la 6.1.0.

Al igual que la última versión desarrollada por fusesource, se trata de un sistema de código abierto, gratuito, que funciona bajo una JVM. En cambio, la licencia pasa a ser LGPL.

Respecto a la última versión de fusesource desarrollada antes de la compra por parte de JBoss, añade tan sólo cambios superficiales como [40]:

- Nueva consola de administración web, más intuitiva y completa.
- Nuevo sistema de corrección de errores.
- Más documentación disponible.

Los requisitos necesarios para su instalación y ejecución son los mismos definidos para el Fuse ESB Enterprise.

4.2.4. Mule ESB

Mule ESB es el sistema ESB diseñado por Mulesoft para arquitecturas orientadas a servicios. Se trata de una de las soluciones junto con las anteriores más utilizadas.

Las principales características de este ESB [41] son:

- *Open source*
- Gratuito durante los treinta primeros días.
- Funciona sobre una JVM.
- Licencia CPAL.

La estructura de este ESB y las funcionalidades que ofrece son las mostradas en la Figura 32.

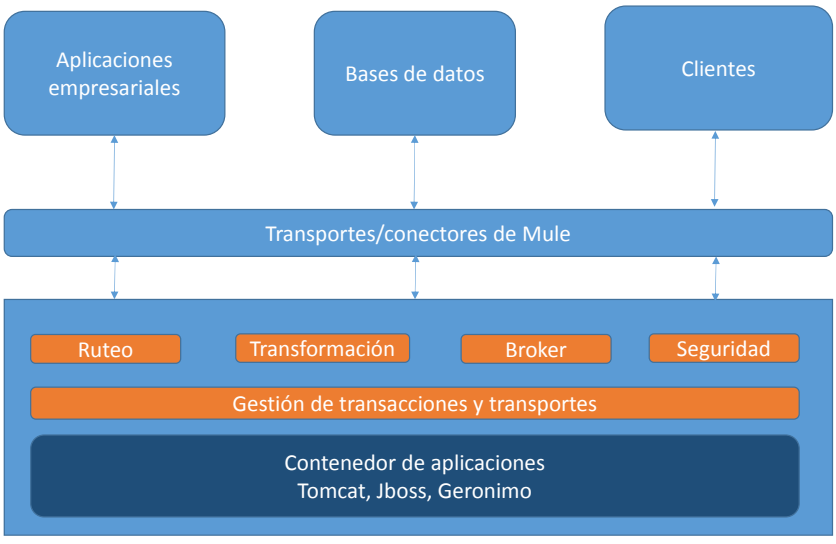

Figura 32. Estructura de Mule ESB.

Como se puede observar, permite interconectar aplicaciones, bases de datos y otros componentes mediante un núcleo de integración de servicios con funcionalidades como enrutamiento, transformación y distribución de mensajes, con requisitos paralelos como sistemas de seguridad y gestión.

Los requisitos mínimos son los siguientes:

- Java 1.6 o superior.
- 2 GiB de memoria RAM.
- Procesador *dual core* de 2 GHz.

4.2.5. Open ESB

Se trata de una plataforma de integración de servicios inicialmente diseñada por Sun Microsystems y Seebeyond, que posteriormente fue adquirido y mantenido por la comunidad Open ESB. Las características principales de este software son:

- Código abierto.
- Licencia CDDL.
- Funciona sobre una JVM.
- Gratuito.

Open ESB [42] se compone de los elementos mostrados en la Figura 33.

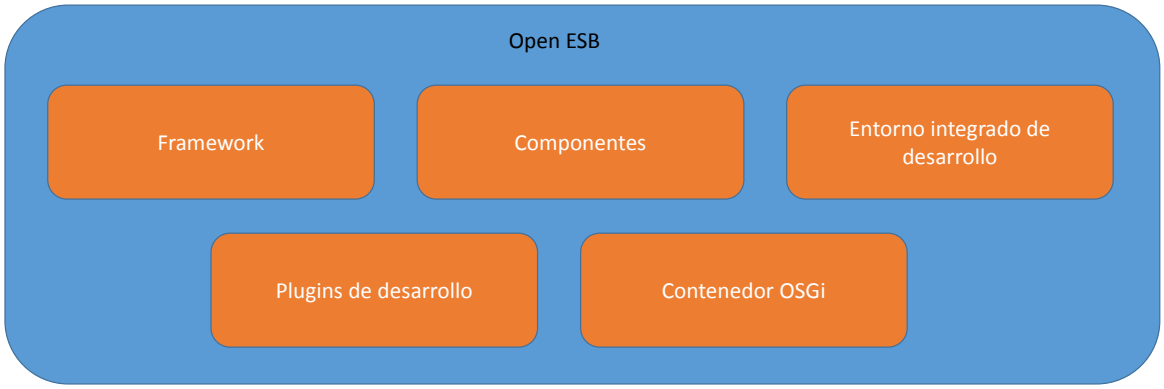

Figura 33. Componentes de Open ESB.

- Un *framework* para la ejecución de aplicaciones y la integración de servicios.
- Componentes para el funcionamiento.
- Entorno de desarrollo integrado.
- *Plugins* para funcionalidades extendidas orientadas a desarrolladores de aplicaciones.
- Contenedores para aplicaciones basadas en OSGi.

Los requisitos mínimos no se especifican, determinando tan sólo la necesidad de poseer una versión de JDK (*Java Development Kit*, o Kit de Desarrollo para Java) superior a la 1.6.

4.2.6. Petals ESB

Petals ESB está desarrollado por la comunidad Petals, bajo el consorcio de middleware OW2. Provee las características y funcionalidades propias de la mayoría de ESB actuales.

Los principales puntos son:

- Código abierto.
- Licencia LGPL.
- Funciona bajo JVM.
- Gratuito.

La estructura general del sistema propuesto por Petals ESB es la mostrada en la Figura 34.

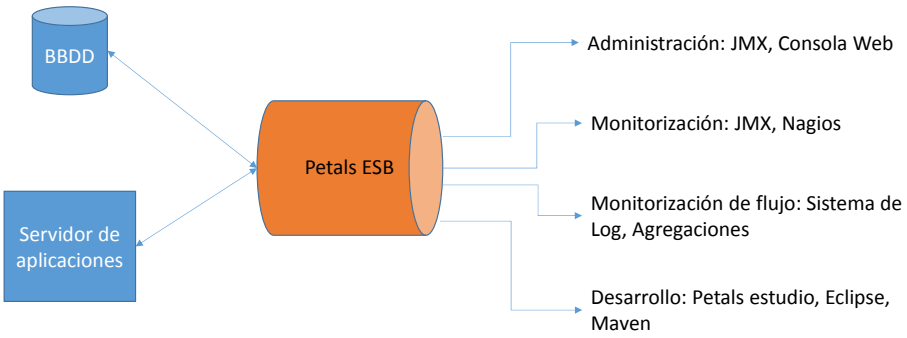

Figura 34. Estructura de Petals ESB.

Como se puede observar, este ESB se encarga de interconectar sistemas heterogéneos, y que provee facilidades para [43]:

- Administración: consolas web y JMX.
- Monitorización: JMX, Negios.
- Monitorización de flujos de mensajes.
- Desarrollo: IDE eclipse, petals estudio.

La última versión de este programa es la 3.0, la cual tiene los siguientes requisitos mínimos:

- Java 1.6 recomendado.
- Procesador de más de 2 GHz.
- 512 MiB de memoria RAM.
- Interfaz de red de 100 Mbit/s.

4.2.7. WSO2 ESB

Es la versión diseñada por WSO2 para una infraestructura orientada a servicios. Soporta una gran cantidad de protocolos y formatos de transporte. Posee los siguientes atributos:

- Código abierto.
- Licencia de Apache.
- Gratuito.

- Funciona bajo JVM.

Entre algunas de sus características, extraídas de la página oficial de WSO2 ESB [44], encontramos:

- Enrutamiento avanzado de mensajes.
- Integración de patrones empresariales o EIP.
- Transformación de mensajes avanzada.
- Consola de administración web y GUI.

Podemos ver esto reflejado en el diagrama que muestra la estructura de este ESB, en la Figura 35.

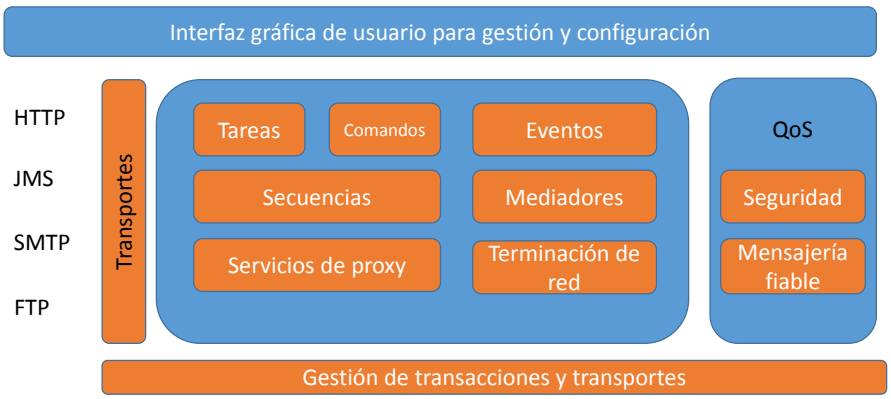

Figura 35. Componentes de WSO2 ESB.

La última versión del ESB es la 4.2.1 tiene los siguientes requisitos mínimos:

- Java Development Kit 1.6
- 512 MiB Memoria RAM
- 125 MiB de espacio en disco duro.

4.2.8. Otras soluciones

Se han visto ya las soluciones más comunes de *Open source* para un ESB. Para no extender el estudio demasiado, se muestran en la Tabla 3 brevemente las características principales de otros ESB menos conocidos.

	Características	Funcionalidades	Requisitos mínimos
Ultra ESB [45]	• JVM • *Open source* • Licencia AGPL • Gratuito	• Mantenimiento y monitorización avanzados • Clustering • Testing avanzado	• JDK 1.6 o superior
Talend ESB [46]	• JVM • *Open source* • Apache software license	• Escalabilidad • Eclipse para el desarrollo • Conectividad extensa	• JDK 1.5 o 1.6

	Características	Funcionalidades	Requisitos mínimos
	• Gratuito		
Membrane ESB [47]	• JVM • *Open source* • Licencia ASF 2.0 • Gratuito	• Despliegue en caliente • Sistemas de seguridad avanzados • Alto rendimiento	• JDK 1.6 o superior

Tabla 3. Soluciones ESB open source menos utilizadas.

4.2.9. Soluciones propietarias

Las soluciones que no son de código abierto están desarrolladas en su mayoría por empresas importantes en el sector, como podrían ser Microsoft o IBM. Estas soluciones poseen licencias propietarias, que son aquellas desarrolladas por las propias empresas, y que definen el marco de utilización del software.

A continuación se definen las soluciones de este tipo más utilizadas en la actualidad, exponiendo sus características principales.

WebSphere ESB

WebSphere es una infraestructura de conectividad flexible para integrar aplicaciones y servicios diseñada por IBM. Se trata de un ESB con las siguientes características [48]:

- Software propietario, diseñado por IBM, y utilización sujeta a licencias propietarias de IBM.
- No es gratuito.
- No se puede comprar el producto por separado, si no que se debe contratar un servicio para el desarrollo de aplicaciones.

WebSphere no funciona con Java, por lo que no es completamente portable, sino que es soportado por los sistemas operativos más comunes, como Windows o sistemas basados en núcleo Linux.

Oracle ESB

Este es el ESB diseñado por la empresa Oracle. Se trata de un ESB con [49]:

- Software propietario, con licencias de Oracle.
- No es gratuito.

Aunque provee las características principales de un ESB, no es completamente uno, ya que no dispone de funcionalidades como enrutamiento de mensajes.

Requisitos mínimos:

- Procesador de al menos 300 Mhz.
- 500 MiB de espacio en disco.
- 2 GiB de memoria RAM.

BizTalk Server

BizTalk Server es la alternativa de Microsoft para el desarrollo de aplicaciones en un entorno orientado a servicios. La última versión de este servidor de aplicaciones es la 2013. Entre sus características encontramos [50]:

- Software propietario, con licencia distribuida por Microsoft.
- No es gratuito.
- Funciona sólo bajo Windows.

Algunas de las funcionalidades avanzadas que ofrece son:

- Integración de aplicaciones empresariales (EAI – *Enterprise Apllication Integration*).
- Automatización de procesos empresariales (BPA – *Business Process Automatization*).
- Modelado de procesos de negocio (BPM – *Business Process Management*).
- Comunicación B2B (*Business-to-business*, de negocio a negocio).
- *Broker* de mensajería

Requisitos mínimos:

- Procesador de 1 GHz o superior.
- 2 GiB de memoria RAM.
- 10 GiB de espacio en disco.
- Windows.

Otras soluciones propietarias

	Requisitos mínimos	Plataforma
Artix ESB	No especificados	SO Windows, Linux
Sonic ESB	No especificados	Java
WebMethods Integration Server	Java 1.4.x	SO Windows, Linux, Solaris

Tabla 4. Soluciones propietarias menos utilizadas.

4.3. Comparativa de las soluciones

Una vez se han visto las diferentes soluciones SOA existentes, es el momento de contraponer sus características y funcionalidades para poder compararlas, y ver cuál de las soluciones se adecúa mejor a los criterios necesarios para su ejecución en el *Raspberry Pi*.

4.3.1. Tabla comparativa

Se muestra en la Tabla 5 una tabla comparativa de cada uno de los productos software evaluados, mostrando su creador, la versión evaluada, y distinguiendo los de código de abierto y los propietarios, cada uno con sus licencias software.

Software	Creador	Versiones	*Open source*	Licencia software
Apache ServiceMix	Apache	4.5.3	Sí	Apache Software license
JBoss Fuse	RedHat	6.1.0	Sí	LGPL
Fuse ESB Enterprise	FuseSource	7.1.0	Sí	Apache Software License
Artix ESB	Micro focus	5.x	No	Propietario
Mule ESB	mulesoft	3.3.1	Sí	CPAL
Open ESB	OpenESB community	2.0	Sí	CDDL
PEtALS ESB	OW2 consortium	3.1.3	Sí	LGPL
Ultra ESB	Adroit Logic	1.6.1	Sí	AGPL
Talend ESB	Talend	5.2	Sí	Apache Software License / Subscription GPL
Membrane ESB	Membrane	4.0.17	Sí	ASF 2.0 *Open source* License
WebSphere ESB	IBM	7.0.0.3	No	Propietario
WSO2 ESB	WSO2	4.7.0	Sí	Apache Software License

Software	Creador	Versiones	*Open source*	Licencia software
Oracle ESB	Oracle	10.1.3.1	No	Propietario
Sonic ESB	Progress Software	8.x	No	Propietario
WebMethods Integration Server	Software AG	9.0	No	Propietario
Biz Talk Server	Microsoft	2013	No	Propietario

Tabla 5. Tabla comparativa de ESB.

4.4. Criterios de elección

Es importante definir los criterios por los cuales se debe distinguir entre los diferentes ESB estudiados para que funcionen bajo el *Raspberry Pi*, y para que cumplan requisitos funcionales y no funcionales del sistema. La nomenclatura es la siguiente:

- RSx: requisito imprescindible del sistema (Requisito Sistema)
- RRx: requisito para el *Raspberry Pi* (Requisito *Raspberry Pi*)
- CAx: criterios adicionales (Criterio Adicional)

En primer lugar, se definen a continuación los requisitos del sistema general:

RS1. **Código abierto**. El entorno elegido debe ser de código abierto, ya que permite la utilización no sólo parcial, si no completa del software elegido. El sistema requerirá el desarrollo de nuevos componentes que se puedan agregar al software para desplegar las funcionalidades propias del sistema.

RS2. **Licencia pública**. Debe tener una licencia pública, de tal forma que permita su completa utilización y distribución.

Una vez definidos los criterios funcionales y no funcionales necesarios en el sistema general, pasamos a definir los necesarios en el controlador local, que en nuestro caso será un *Raspberry Pi*. Estos son los siguientes:

RR1. **JVM**. Los anteriores requisitos se resumen en la posibilidad de ejecutar el ESB bajo una JVM. De esta manera, tendrán preferencia los ESB que funcionan bajo Java.

RR2. **JVM 1.6**. La versión de Java necesaria para ejecutar el ESB debe ser igual o inferior a la 1.6, que es la máxima soportada por Raspbian. Se tendrá preferencia por los ESB que se ejecuten especialmente en la versión 1.6, que es la más pulida para *Raspberry Pi*.

RR3. **Hardware**. Los requisitos mínimos deben ser lo menos restrictivos posibles, ya que el *Raspberry Pi* tiene una capacidad de procesamiento muy limitada.

RR4. **Instalación**. Aunque se cumplan todos los requisitos, es necesario ver que la instalación en el *Raspberry Pi* se realiza con normalidad. Por ello, los ESB que se instalen sin ningún tipo de problema serán los que cumplan este requisito.

Para poder distinguir entre soluciones que satisfagan las anteriores especificaciones, se tendrán en cuenta aspectos como:

CA1. **Versiones mínimas**. Versiones mínimas del programa, al cual se le puedan añadir los módulos que necesitamos necesariamente, sin tener que añadir otros que no se utilizarán.
CA2. **Documentación**. Presencia de documentación amplia sobre el programa.
CA3. **Otros**. Funcionalidades como:
- Administración avanzada.
- Herramientas para desarrolladores.
- Entorno de desarrollo integrado.

4.5. Alternativa elegida

En este apartado se justifica la elección tomada de acuerdo al ESB escogido de acuerdo a los criterios que se mencionan en el apartado anterior. Asimismo, se detalla el funcionamiento de la alternativa elegida y la estructura de forma mucho más amplia que en el apartado 4.1.

4.5.1. Justificación de la elección

En la Tabla 6 se pueden observar el cumplimiento de los requisitos y criterios anteriormente mencionados. En ella se contraponen dichos criterios a cada uno de los ESB estudiados.

✓✖	RS1	RS2	RR1	RR2	RR3	RR4	CA1	CA2	CA3
ServiceMix	✓	✓	✓	✓	✓	✓	✓	✓	✓
Fuse ESB Enterprise	✓	✓	✓	✓	✓	✓	✖	✓	✓
JBoss Fuse	✓	✓	✓	✓	✓	✓	✖	✓	✓
Mule ESB	✓	✓	✓	✓	✓	✖	✖	✓	✓
Open ESB	✓	✓	✓	✓	✓	✖	✖	✖	✖
Petals ESB	✓	✓	✓	✓	✓	✖	✖	✓	✖
WSO2 ESB	✓	✓	✓	✓	✓	✖	✖	✓	✓
Ultra ESB	✓	✓	✓	✓	✓	✖	✖	✖	✖
Talend ESB	✓	✓	✓	✓	✓	✖	✖	✖	✖
Membrane ESB	✓	✓	✓	✓	✓	✖	✖	✖	✖
Artix ESB	✖	✖	✓	✖	✖	✖	✖	✓	✓

✓✖	RS1	RS2	RR1	RR2	RR3	RR4	CA1	CA2	CA3
WebSphere ESB	✖	✖	✓	✖	✖	✖	✖	✓	✓
Oracle ESB	✖	✖	✓	✓	✓	✖	✖	✓	✓
Sonic ESB	✖	✖	✓	✖	✖	✖	✖	✓	✓
WebMethods Integration Server	✖	✖	✓	✖	✖	✖	✖	✓	✓
BizTalk Server	✖	✖	✓	✖	✖	✖	✖	✓	✓

Tabla 6. Tabla de cumplimiento de requisitos.

A partir de estos datos, se pueden establecer los siguientes puntos:

- Primero, se han descartado los ESB que no cumplen los requisitos RSx. De acuerdo a estos requisitos se han descartado los siguientes ESB:
  - Artix ESB
  - WebSphere ESB
  - Oracle ESB
  - Sonic ESB
  - WebMethods Integration Server
  - BizTalk Server
- En Segundo lugar, se descartan los ESB que no cumplen los requisitos RRx:
  - Mule ESB
  - Open ESB
  - Petals ESB
  - WSO2 ESB
  - Ultra ESB
  - Talend ESB
  - Membrane ESB
- Por último, y con los ESB resultantes, se evalúan los criterios CAx. De esta forma, los ESB resultantes son:
  - JBoss fuse
  - Fuse ESB Enterprise
  - Apache ServiceMix

De estos tres ESB, se ha decidido elegir la alternativa propuesta por Apache con su ServiceMix en base a las siguientes razones:

- JBoss Fuse y Fuse ESB se basan en ServiceMix, al cual se le añaden nuevas funcionalidades. Por tanto, si estas funcionalidades no son necesarias, proporcionarán una carga de trabajo mayor al dispositivo, cuando lo que se busca es no sobrecargar el sistema.

- ServiceMix posee una versión mínima, a la cual podremos añadir únicamente los módulos necesarios.

Por tanto, la elección final es Apache ServiceMix 4.5.3, que cumple los siguientes requisitos:

- Sistema de código abierto.
- Licencia pública de Apache.
- Único ESB evaluado con versión *minimal*, que permite instalar módulos según se necesiten.
- Interoperabilidad con Fuse ESB, ya que se basa en ServiceMix para su funcionamiento, añadiendo solo algunas funcionalidades.
- No tiene requisitos mínimos de hardware, exceptuando el tamaño que conlleve su instalación.
- Permite su funcionamiento bajo Raspbian, ya que utiliza una JVM para su funcionamiento.
- Es sólo compatible con Java JDK 1.6, que es precisamente la versión más refinada que utiliza el *Raspberry Pi*.
- Al probar su instalación, habiendo establecido los requisitos previos necesarios, no hubo ningún problema en su ejecución, ni en la ejecución de ejemplos de aplicaciones y servicios.
- Existe numerosa documentación online, e incluso libros escritos para comprender su funcionamiento, y para obtener guías de desarrollo de servicios y aplicaciones.
- Entre otras funcionalidades existentes, destacan por ejemplo:
    - Consola de administración Web avanzada.
    - Sistema automatizado de logs.
    - Despliegue de *bundles* en caliente.
    - Herramientas avanzadas de gestión y monitorización.
    - Acceso remoto mediante SSH.

4.5.2. Funcionamiento y componentes

Apache ServiceMix [38] es un ESB desarrollado a partir de otros proyectos software de Apache. Dispone de diferentes módulos que ofrecen la funcionalidad completa del software. Para la versión *minimal*, será necesario disponer de los siguientes módulos:

Apache Felix: es una implementación de código abierto de la especificación de OSGi versión 4. Proporciona un motor completo para la ejecución de aplicaciones y módulos basados en OSGi, estando integrado dentro de ServiceMix.

Apache Karaf: Karaf se trata de una iniciativa de Apache que se basa en un contenedor para OSGi, al que se le añaden algunas nuevas funcionalidades, como se puede ver en la Figura 36.

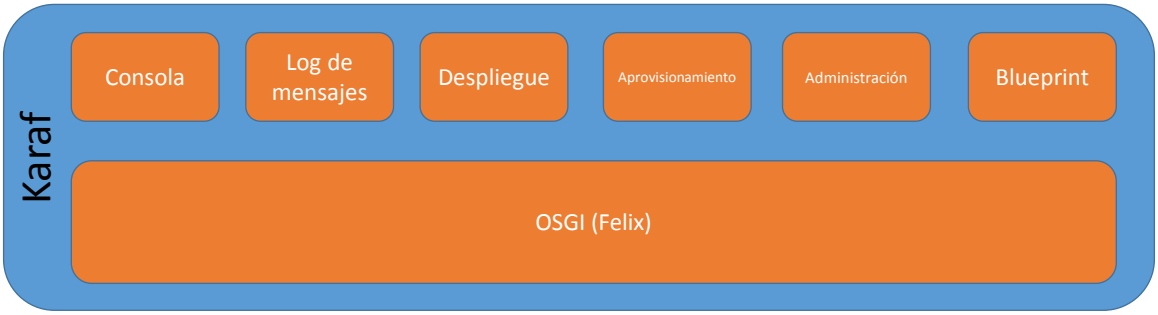

Figura 36. Kernel de Apache Karaf.

Se observa que Karaf utiliza Felix como motor OSGi, al que le añade funcionalidades como:

- Consola de administración.
- Log de mensajes.
- Despliegue de aplicaciones.
- Interfaz de administración.
- Uso de contenedores *Blueprint* y *spring DM.*

En la versión 4.5.3 de apache ServiceMix se utiliza la versión 2.2.11 de Apache Karaf.

ActiveMQ: ActiveMQ (Active Message Queing) se trata de un sistema de mensajería, que implementa arquitecturas MOM (*Message Oriented Middleware*) dentro de arquitecturas orientadas a servicios. Para ello, implementa la especificación *Java Message Service* (JMS), así como otros protocolos de mensajería entre aplicaciones como AMQP. En la versión 4.5.3 de Apache ServiceMix, se utiliza la versión 5.7.0 de ActiveMQ.

CXF: CXF es un entorno diseñado por Apache para la inclusión de *Web Service*s, que posee las siguientes características:

- Separación entre *endpoints*.
- Simplicidad de creación de clientes y *endpoints*.
- Alto rendimiento sin sobrecargas.
- Empotrable dentro de otros proyectos de Apache.

Entre algunas de las tecnologías que soporta, se encuentran:

- SOAP
- *WS Addressing, Policy, ReliableMessaging, SecureConversation, Security* y *SecurityPolicy*.
- CORBA (*Common Object Request Broker Architecture*).
- HTTP y JMS
- REST
- Despliegue empotrado en OSGi, Tomcat o ServiceMix.

Blueprint y Spring: Tanto Blueprint como Spring definen formas para encapsular *bundles* desarrollados para su ejecución en una especificación OSGi. Permiten la publicación de interfaces como servicios, exposición de URIs para servicios WS, enrutamiento mediante sintaxis XML, y otras muchas funcionalidades.

La versión a utilizar de este ESB es una versión mínima. Por ello, no dispone de los componentes mencionados, exceptuando el necesario motor OSGi. Los componentes que se instalarán serán:

- CXF para el desarrollo de la interfaz REST mencionada en el capítulo 5 de este documento.
- ActiveMQ, para proveer de comunicación mediante JMS al controlador de dispositivos.

5. Desarrollo del controlador de dispositivos

En el capítulo 3 de esta memoria se redactan las funcionalidades que lleva a cabo un Controlador de dispositivos. El desarrollo de un controlador de dispositivos, completamente funcional, permitirá evaluar el rendimiento del ESB Apache ServiceMix elegido de acuerdo a las conclusiones del apartado anterior. De esta forma, se demostrará si efectivamente, y como sugiere el título de este proyecto, el *Raspberry Pi* puede ser un soporte válido como herramienta de despliegue

En este apartado se definirán las diferentes herramientas y soportes utilizados para el desarrollo del Controlador de dispositivos. Asimismo, se expondrán tanto el diseño y la arquitectura software, como su funcionamiento e intercambio de información.

5.1. Herramientas utilizadas y requisitos previos

Para el desarrollo del Controlador de dispositivos, es necesario poseer un entorno específico que nos permita compilar y ejecutar el proyecto, así como desarrollar el código necesario. En los siguientes apartados se definen cada una de estas herramientas: su funcionalidad, y su forma de funcionamiento.

5.1.1. Java JDK

Apache ServiceMix corre sobre la JVM de Java. Por lo tanto, para poder ejecutar el ESB, es necesario tener un JDK instalado. Apache ServiceMix necesita una versión del JDK de Java igual o inferior a la 1.6.

Para su instalación en el *Raspberry Pi*, se ha utilizado la versión del JDK diseñada por Open JDK en vez de la oficial de Oracle, ya que esta última tiene un menor soporte actualmente en Raspbian.

Por otro lado, una vez instalado el JDK, es necesario crear las variables de entorno necesarias para que ServiceMix detecte la versión actual de Java.

La instalación del JDK y la creación de las variables de entorno se definen en el manual de usuario del proyecto.

5.1.2. Apache Maven

Apache Maven es una herramienta para el desarrollo y gestión de proyectos Java que permite realizar de forma automática tareas generales como la creación de los directorios de nuestro proyecto, la compilación, generación de paquetes y documentación entre otros. Sin embargo, la principal característica es la posibilidad de añadir módulos externos necesarios mediante dependencias [51].

Maven descarga de forma automática las dependencias necesarias anotadas en el proyecto a partir de los repositorios online y crea un repositorio local con los módulos descargados. De esta forma, es muy cómodo añadir importaciones de otros módulos software, sin necesidad de descargar cada uno de ellos y almacenarlo de forma local.

En el repositorio central de Maven se puede encontrar cada una de estas dependencias. No obstante, también se pueden añadir dependencias de módulos que no se encuentren en el repositorio online.

Para este proyecto se utiliza la versión 2.0 de Maven. Su instalación y la creación de las variables de entorno necesarias vienen definidas en el manual de usuario.

5.1.2.1. *Project Object Model*

La herramienta Maven utiliza el concepto de POM (*Project Object Model* o Modelo de Objetos de Proyecto) para la realización de todas estas tareas. Mediante un archivo con una sintaxis XML se pueden definir las dependencias del proyecto con otros módulos, las opciones de compilación, y el uso de *plugin*s externos para otras funciones específicas.

Este archivo XML tiene una estructura como la siguiente, obtenida de la página oficial de Maven [52]:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                          http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <!-- The Basics -->
  <modelVersion>...</modelVersion>
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <packaging>...</packaging>
  <dependencies>...</dependencies>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagement>
  <modules>...</modules>
  <properties>...</properties>


  <!-- Build Settings -->
  <build>...</build>
  <reporting>...</reporting>

  <!-- More Project Information -->
  <name>...</name>
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
```

```
  <organization>...</organization>
  <developers>...</developers>
  <contributors>...</contributors>

  <!-- Environment Settings -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

De todos estos elementos pertenecientes al archivo de definición del POM, se definen a continuación los elementos estrictamente necesarios para la creación del proyecto.

Elementos de definición del proyecto

- **groupId**: define un identificador único para un grupo de proyectos determinado.
- **artifactId**: este identificador define unívocamente al proyecto que se genera.
- **version**: permite especificar la versión del proyecto actual.

Definición de relaciones y dependencias

Las diferentes dependencias con otros módulos y las relaciones se definen en el elemento <dependencies>, que contiene un número determinado de elementos <dependency> con la siguiente estructura:

```
<dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.0</version>
      <type>jar</type>
      <scope>test</scope>
      <optional>true</optional>
</dependency>
```

- **groupId**, **artifactId**, **version**: como antes, identifican a un módulo determinado.
- **type**: define el tipo de empaquetación de la dependencia.
- **scope**: define el radio de uso de la dependencia definida. Si no se especifica, se supone el uso en tiempo de compilación y ejecución. Puede tener los siguientes valores:
    - **compile**: sólo se utiliza la dependencia en tiempo de compilación.

- **provided**: sólo se utiliza la dependencia en tiempo de compilación y de ejecución.
- **runtime**: sólo se utiliza la dependencia en tiempo de ejecución.
- **test**: sólo se utiliza la dependencia al realizar los tests.
- **system**: se utiliza la dependencia en el contexto general del sistema.
- **optional**: define si es una dependencia estrictamente necesaria o no.

Elementos de construcción

La construcción del proyecto se realiza de acuerdo a las diferentes reglas descritas dentro del elemento <build>. Aquí también se definen los diferentes *plugins* que se utilizan, así como otros elementos tales como recursos utilizados, o gestión de los propios *plugins*.

El elemento `<build>` es un elemento opcional. No obstante, en la mayoría de los proyectos es necesario definir algunas propiedades dentro de este elemento para la compilación del proyecto.

5.1.2.2. *Arquetipos y estructura de directorios*

Maven dispone de unas plantillas, llamadas arquetipos, que permiten crear tipos de proyectos predefinidos. De esta forma, se puede crear una estructura de directorios y ficheros determinada para un proyecto determinado. Además, también pueden incluir clases genéricas que modelan un tipo de proyecto determinado.

En la Figura 37 se muestra una tabla extraída de la página oficial de Maven con los arquetipos más comunes [53].

Archetype ArtifactIds	Description
maven-archetype-archetype	An archetype which contains a sample archetype.
maven-archetype-j2ee-simple	An archetype which contains a simplifed sample J2EE application.
maven-archetype-mojo	An archetype which contains a sample a sample Maven plugin.
maven-archetype-plugin	An archetype which contains a sample Maven plugin.
maven-archetype-plugin-site	An archetype which contains a sample Maven plugin site.
maven-archetype-portlet	An archetype which contains a sample JSR-268 Portlet.
maven-archetype-quickstart	An archetype which contains a sample Maven project.
maven-archetype-simple	An archetype which contains a simple Maven project.
maven-archetype-site	An archetype which contains a sample Maven site which demonstrates some of the supported document types like APT, XDoc, and FML and demonstrates how to i18n your site.
maven-archetype-site-simple	An archetype which contains a sample Maven site.
maven-archetype-webapp	An archetype which contains a sample Maven Webapp project.

Figura 37. Tipos de arquetipos de Maven

De esta forma, es mucho más sencilla la creación de un proyecto con un patrón determinado, ya que no es necesario crear las clases y la estructura de directorios y ficheros de forma manual.

Como se puede observar en la tabla anterior, los arquetipos se identifican con un identificador similar al usado para describir las dependencias en un proyecto.

En este proyecto, se utilizará el arquetipo *maven-archetype-quickstart*. Este arquetipo crea un proyecto genérico de Maven. La estructura de directorios se mostrará en el apartado de desarrollo.

En apartados posteriores se especificará el contenido de los directorios y su funcionalidad.

5.1.2.3. *Repositorio*

El repositorio de Maven se crea y se genera automáticamente a partir de las dependencias indicadas en un proyecto. De esta forma, no se necesita importar local y manualmente cada uno de los módulos software de los que depende nuestro proyecto.

Dicho repositorio se crea en la carpeta .m2 perteneciente al directorio principal del usuario.

5.1.2.4. *Comandos*

Para la creación, compilación, y otras funciones como la realización de pruebas, Maven debe ser ejecutado desde una consola o terminal del sistema operativo. En el manual de usuario se define el procedimiento general para la creación de un proyecto y la aplicación de las operaciones que se pueden realizar sobre él, así como los diferentes comandos y salida de la consola

5.1.3. Eclipse

Eclipse es un entorno de desarrollo para la creación de proyectos de programación, conteniendo un entorno de desarrollo integrado para plataformas como Java.

Aunque no es un elemento indispensable, su utilización se antoja necesaria, ya que permite corregir los errores de compilación y desarrollar y depurar código de una forma sencilla.

Para el desarrollo del proyecto, se ha utilizado la versión Kepler del programa, que contiene herramientas para el desarrollo de aplicaciones Java EE. Entre esas herramientas, se encuentra un *plugin* para la creación de proyectos Maven.

5.1.3.1. *Maven plugin*

Con el *plugin* de Maven, se puede realizar un proyecto Maven desde el IDE Eclipse. La instalación de este *plugin* no es necesaria en el caso de la versión de Eclipse orientada a desarrolladores de Java EE. Sin embargo, se adjunta en el manual de usuario un manual de instalación de dicho *plugin*.

Con este *plugin* es posible realizar funciones como:

- Creación de nuevos proyectos, tanto genéricos como mediante arquetipos.
- Generación automática de ficheros de despliegue, como Blueprint o POM.
- Compilación y ejecución de módulos.
- Ejecución de comandos de Maven de forma automática.

5.2. Estructura de directorios y archivos necesarios

La estructura de directorios y el proyecto se han creado a partir de un arquetipo Maven. Dicho arquetipo es *maven-archetype-quickstart*, y crea una estructura como la de la Figura 38.

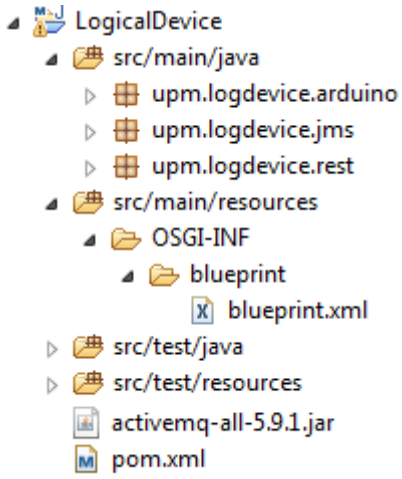

Figura 38. Estructura de directorios y ficheros del proyecto.

- **src**: contiene los archivos que definen el proyecto.
  - **main/Java:** contiene el código del programa.
  - **main/resources:** contiene los archivos necesarios para la creación del budle.
    - **OSGI-INF/Blueprint:** aquí está contenido el archivo *Blueprint.xml*, que establece la forma en que se crea el contenedor Blueprint para el *bundle*.
- **test:** contiene los archivos necesarios para ejecutar los test previos.
- **target**: contiene los archivos compilados del proyecto. Aquí se encuentra el archivo *logDevice-0.0.1-SNAPSHOT.jar*, que es el *bundle* del Controlador de Dispositivos que se desplegará en el ESB.
- **pom**.xml: contiene la estructura del proyecto.
- **activemq**-all-5.9.1.jar: contiene el paquete con las clases necesarias para JMS.

La estructura del POM es la indicada en el fichero pom.xml, que contiene lo siguiente:

```
<project   xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>upm.smart.device</groupId>
  <artifactId>LogDevice</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Controlador de dispositivos</name>
  <description>Controlador de dispositivos</description>

    <dependencies>
      <dependency>
        <groupId>org.apache.felix</groupId>
        <artifactId>org.osgi.core</artifactId>
        <version>1.0.0</version>
      </dependency>
```

```
      <dependency>
         <groupId>org.apache.ServiceMix.specs</groupId>
         <artifactId>org.apache.ServiceMix.specs.jsr311-api-1.1.1</artifactId>
         <version>1.9.0</version>
         <scope>provided</scope>
      </dependency>

   </dependencies>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <extensions>true</extensions>
            <configuration>
               <instructions>
                  <Bundle-Name>controlador de dispositivos </Bundle-Name>
                  <Bundle-SymbolicName>${pom.artifactId}</Bundle-SymbolicName>
                  <Import-Package>*</Import-Package>
               </instructions>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

En este proyecto se han definido dos dependencias necesarias para la ejecución del *bundle*:

- La primera dependencia se corresponde con los módulos necesarios para ejecutar un *bundle* OSGi en ServiceMix.
- La segunda dependencia permite el uso de CXF, que es una implementación de Apache contenida en el ServiceMix para el uso de *Web Service*s. En este caso, se corresponde con la especificación de un *Web Service* de tipo RESTful.

Por otra parte, se incluye en el apartado *build* un *plugin*, que tiene como objetivo la creación y compilación del *bundle* OSGi mediante el *plugin* de Maven contenido en el IDE de Eclipse.

Para poder encapsular el *bundle* desarrollado en un contenedor OSGi mediante Blueprint, se utiliza el fichero Blueprint.xml contenido en la carpeta /src/main/resources/OSGI-INF/Blueprint. Tiene la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<Blueprint xmlns="http://www.osgi.org/xmlns/Blueprint/v1.0.0"
```

```
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jaxrs="http://cxf.apache.org/Blueprint/jaxrs"
   xsi:schemaLocation="
              http://www.osgi.org/xmlns/Blueprint/v1.0.0
http://www.osgi.org/xmlns/Blueprint/v1.0.0/Blueprint.xsd
              http://cxf.apache.org/Blueprint/jaxrs
http://cxf.apache.org/schemas/Blueprint/jaxrs.xsd">


     <bean  id="jmsInterface"  class="upm.logdevice.jms.Launcher"  init-method="init"
destroy-method="destroy"/>

     <jaxrs:server id="logicalDevice" address="/logDevice">
        <jaxrs:serviceBeans>
            <ref component-id="restAccess"/>
        </jaxrs:serviceBeans>
    </jaxrs:server>

    <bean id="restAccess" class="upm.logdevice.rest.AccessData"/>

</Blueprint>
```

Mediante el elemento *bean id=jmsInterface* se define la clase que ejecutará el método *init* con las operaciones requeridas al arrancar el *bundle* para la comunicación mediante JMS.

Para poder realizar peticiones via REST al *bundle*, se establece una URI, mediante el elemento *jaxrs:server*, que define la URI */logDevice* para acceder a las operaciones disponibles via REST. Estas operaciones se referencian en el servidor mediante el *bean restAccess*, definido en el elemento *ref*.

5.3. Diseño del Controlador de Dispositivos

En este apartado se define el diseño del Controlador de dispositivos, especificando los componentes software que lo conforman, y el desarrollo y las funcionalidades de cada uno de los objetos que conforman dichos componentes.

En la Figura 39 se muestran los diferentes paquetes que conforman el proyecto, y las clases contenidas en cada uno de ellos:

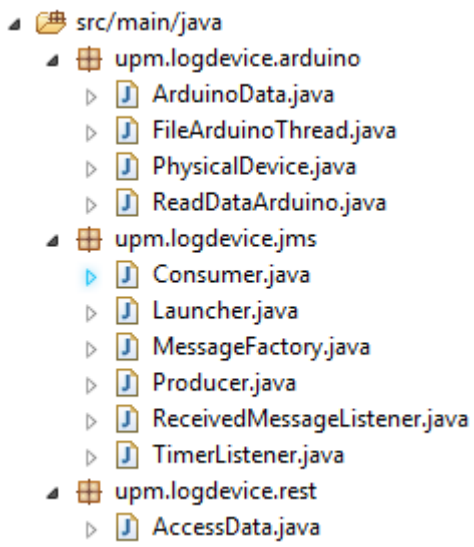

Figura 39. Paquetes y clases de Java del proyecto.

A continuación se define cada uno de estos módulos, describiendo el esquema de funcionamiento, y la funcionalidad de cada una de las clases contenidas en dicho paquete.

5.3.1. Paquete Arduino

El módulo de Arduino contiene las clases necesarias para el acceso y la comunicación con el dispositivo físico Arduino conectado al *Raspberry Pi*. Provee una interfaz para el acceso a los datos ofrecidos por los sensores del dispositivo físico.

La comunicación con el dispositivo Arduino que recoge las medidas del resto de sensores basados en Arduino se realiza mediante una conexión USB directa al *Raspberry Pi*. La ventaja es que Raspbian detecta dicha conexión como un puerto Serie, y puede ser accedido mediante Java como si se tratara de un fichero.

Para ello, existen dos formas de acceso:

1. Volcar la información ofrecida por los sensores en un fichero.

2. Acceder a la última lectura ofrecida a través del puerto serie, tratando su acceso como el acceso a un fichero.

En el manual de usuario se detalla cada uno de estos dos procesos.

Existen cuatro clases que conforman este módulo:

- **Arduino*Data*:** modela la información devuelta por uno de los sensores Arduino, incluyendo valores como el valor ofrecido, las unidades de medida y la identificación del sensor.
- ***PhysicalDevice***: contiene los parámetros necesarios para identificar a un dispositivo físico: el fabricante, el número de serie, y el identificador de modelo.
- ***File*Arduino*Thread***: esta clase implementa las operaciones realizadas por un hilo que tiene como objetivo borrar las medidas antiguas que se almacenan en el fichero sobre el que se vuelcan cada

segundo las medidas ofrecidas por los sensores. Si se lee directamente desde el puerto Serie, este hilo nunca se activará, ya que puede generar errores.

- ***ReadData*Arduino**: esta clase contiene los métodos necesarios para recabar la información de los sensores. En el manual de usuario se detalla las formas de acceso a los datos de los sensores.

5.3.2. Paquete JMS

Este es el componente principal del Controlador de dispositivos, y es el que permite la comunicación con el componente de la infraestructura *e-GOTHAM* mediante JMS.

La implementación de JMS es llevada a cabo mediante ActiveMQ, que proporciona servicio de mensajería entre aplicaciones. El funcionamiento y el objetivo de éste se definen en el apartado 3, en el cual se evalúa el ESB Apache ServiceMix, y en el que se definen los componentes de dicho ESB.

Tres componentes abstractos son los que configuran el funcionamiento de este módulo, y son los siguientes:

- ***Broker*:** el *Broker* es el encargado de recibir y enviar los mensajes generados por la aplicación, creando las conexiones necesarias.
- **Productor**: el productor es el encargado de la generación de los mensajes. Mediante este componente podemos establecer las colas a las que dirigir un mensaje, y los parámetros específicos que necesita dicho mensaje.
- **Consumidor**: el consumidor es el encargado de la recepción de los mensajes en una determinada cola de mensajes de entrada. Con este componente podemos realizar las acciones indicadas para un mensaje de entrada determinado.

Estos componentes abstractos se interrelacionan de la manera definida en el esquema de la Figura 40.

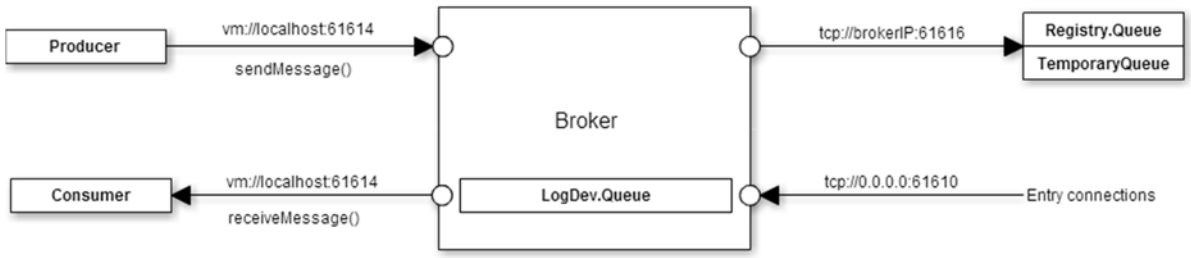

Figura 40. Interconexión de productor, consumidor y Broker.

A continuación se definen las funcionalidades de cada una de las clases:

- ***Launcher***: mediante esta clase, podemos inicializar el *Broker* de mensajería y el productor y consumidor de mensajes.
- ***Consumer***: esta clase modela al consumidor de mensajes. El consumidor se encarga de recibir y despachar los mensajes recibidos. En la implementación, el consumidor obtendrá los mensajes de la cola llamada *LogDev.Queue*.
- ***Producer***: esta clase modela al productor de mensajes. El productor generará los mensajes con destino el *Broker* del controlador central. Utiliza dos colas para el envío de los mensajes:

  - *Registry.Queue:* a esta cola se envían los mensajes que tienen como función el registro del dispositivo y el envío de mensajes para el protocolo de *keep-alive*.
  - *TemporaryQueue:* es una cola temporal y única entre el Controlador de dispositivos y el *e-GOTHAM* sobre la que recibir los mensajes de respuesta a una petición de datos.
- ***MessageFactory***: esta clase provee métodos que contienen las plantillas para la creación de los mensajes intercambiados a partir de los parámetros necesarios, y también disponen de métodos para la lectura de parámetros necesarios de los mensajes recibidos.
- ***ReceivedMessageListener***: mediante esta clase, se realizan las acciones indicadas para cada uno de los tipos de mensajes que pueden ser recibidos por el Controlador de dispositivos. ***TimerListener***: esta clase crea un *timer* que se activa para generar el envío de mensajes periódicos, necesarios para el protocolo de *keep-alive*.

En la Figura 41 se puede ver como se interrelacionan las diferentes clases.

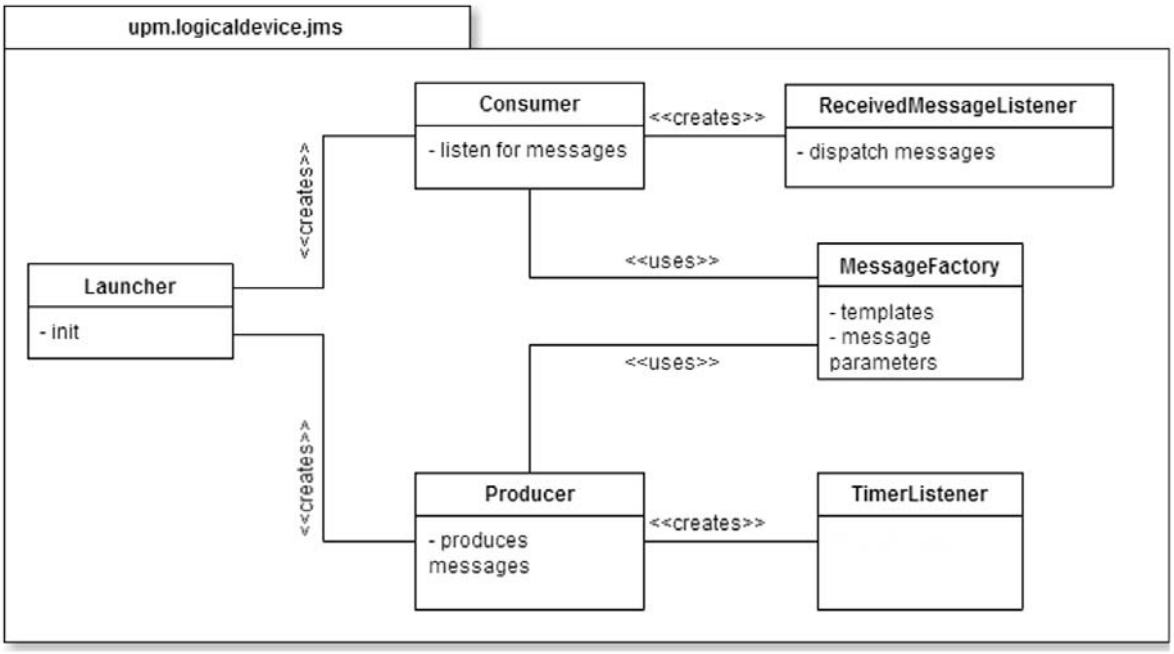

Figura 41. Clases y su interacción en el módulo de JMS.

5.3.3. Paquete REST

El módulo de REST permite utilizar una interfaz RESTful para el acceso a los datos recogidos mediante los sensores Arduino. Este módulo dispone de una sola clase llama RestAccess, que se encarga de implementar:

- La recogida de los valores de los sensores.
- Creación de la respuesta HTML.

- Exposición de las URIs para cada una de las operaciones. Para ello, se genera un WSDL que permite acceder a las operaciones disponibles. Podemos acceder a él una vez iniciado el *bundle*, ingresando la siguiente dirección:

  http://[IPRaspberry Pi]:8181/cxf

Aquí se encuentran los servicios que se encuentran expuestos. Podemos observar cómo se expone el servicio REST del Controlador de dispositivos en la Figura 42.

Figura 42. Descripción del servicio web.

Si se hace *click* sobre el WADL (*Web Application Description Language*, o Lenguaje de Descripción de Aplicativos Web), se obtiene el siguiente documento:

```
<application xmlns="http://wadl.dev.Java.net/2009/02"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <grammars/>
   <resources base="http://localhost:8181/cxf/logDevice">
      <resource path="/Arduino">
         <resource path="/getRandom">
            <method name="GET">
               <response>
                  <representation mediaType="text/html"/>
               </response>
            </method>
         </resource>
         <resource path="/getCurrent">
            <method name="GET">
               <response>
                  <representation mediaType="text/html"/>
               </response>
            </method>
         </resource>
      </resource>
   </resources>
</application>
```

Para acceder por tanto a cada una de las operaciones, se deberán ingresar las siguientes URIs:

- getCurrent: *http://[IPRaspberry Pi]:8181/cxf/logDevice/*Arduino*/getCurrent*
- getRandom: *http://[IPRaspberry Pi]:8181/cxf/logDevice/*Arduino*/getRandom*

5.4. Funcionalidad del controlador de dispositivos

La funcionalidad general del controlador de dispositivos está descrita en la arquitectura *e-GOTHAM* y define las diferentes operaciones llevadas a cabo mediante el intercambio de mensajes en formato XML entre el Controlador de dispositivos y el controlador central a través de una interfaz REST. Las operaciones implementadas se resumen en:

- **Operaciones de Registro de dispositivos**. Mediante el procedimiento de registro, el controlador de dispositivos se registra dentro de la arquitectura de *e-GOTHAM* para que pueda recibir las peticiones de datos realizadas desde dicha arquitectura.
- ***Keep-alive***. Se implementa un protocolo de *keep-alive*, de tal forma que el sistema conoce si un controlador de dispositivos está activo.
- **Petición y respuesta de datos**. Una vez que el controlador de dispositivos ha sido registrado, se pueden empezar a realizar peticiones para recoger datos ofrecidos por los sensores.
- **Parada de ejecución** En caso de parar el controlador de dispositivos de forma local, se debe informar de la circunstancia al controlador central.

5.4.1. Interfaz REST

Mediante la interfaz REST diseñada, se puede acceder a las operaciones disponibles para los dispositivos físicos, para obtener valores ofrecidos por los sensores basados en Arduino.

Para acceder a cada uno de estos servicios, se utilizan las URIs que previamente se han explicado. Si se accede desde un navegador, se podrá ver una salida en forma HMTL de la forma mostrada en la Figura 43.

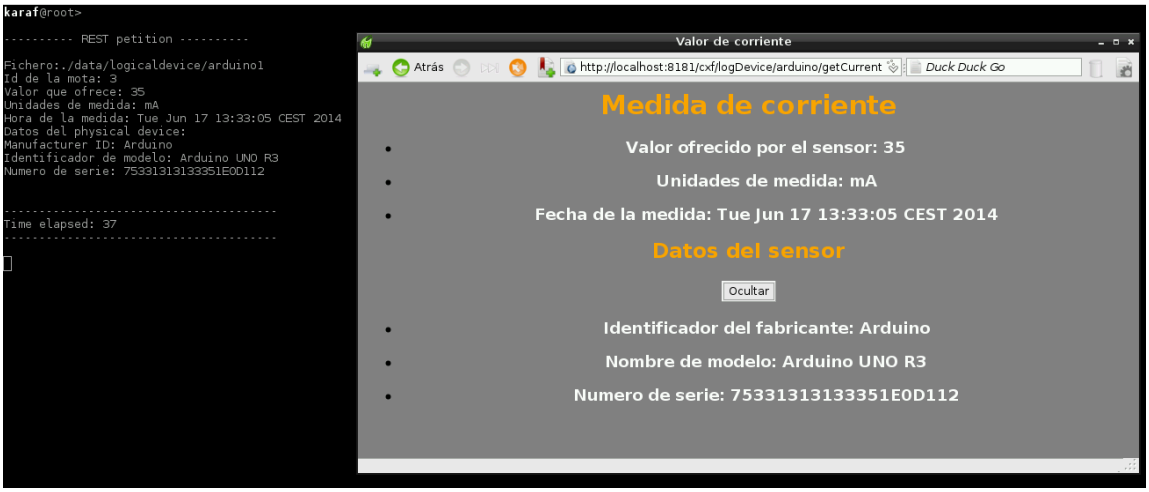

Figura 43. Obtención de valor mediante interfaz web.

6. Pruebas y rendimiento

El proceso final para poder evaluar al *Raspberry Pi* como posible elemento de despliegue de servicios en el sistema planteado, es analizar el comportamiento del software diseñado para el controlador de dispositivos desarrollado en el ESB elegido para el dispositivo.

En este apartado se realizarán mediciones de tiempo que indicarán el rendimiento real del *Raspberry Pi*, y se compararán los resultados obtenidos con la misma configuración software en un ordenador personal.

Después de las pruebas, se analizarán los resultados obtenidos para poder concluir si el *Raspberry Pi* puede ser una herramienta efectiva para el despliegue de servicios.

6.1. Entorno y pruebas

En este apartado se definen las diferentes pruebas que se realizan en el sistema, así como el entorno específico en el que se realizan estas pruebas.

6.1.1. Entorno

Para poder realizar una comparación de rendimiento, se han establecido dos configuraciones hardware, definidas en las tablas Tabla 7 y Tabla 8. En cada una de las dos se ha instalado el ESB elegido en el capítulo 4.

Configuración 1. PC	
Sistema	PC
CPU	Intel Core 2 Duo E6750, doble núcleo a 2.33 ghz.
Memoria RAM	3 GiB DDR2-667 Mhz
Almacenamiento	SATAII 320 GiB
Red	10/100 Mbit Ethernet
Sistema operativo	Windows 7 x86 (32 bit)
ESB	Apache ServiceMix 4.5.3 minimal. • ActiveMQ 5.7.0 • CXF-JAXRS

Tabla 7. Especificaciones de la configuración 1.

Configuración 2. *Raspberry Pi*	
Sistema	*Raspberry Pi*
CPU	ARM 700 MHz
Memoria RAM	512 MiB SDRAM (*Synchronous Dynamic Random Access Memory*, o Memoria de Acceso Aleatoria Dinámica y Síncrona)
Almacenamiento	Tarjeta SDHC Clase 10 de 16 GiB
Red	10/100 Mbit Ethernet
Sistema operativo	Raspbian
ESB	Apache ServiceMix 4.5.3 minimal • ActiveMQ 5.7.0 • CXF-JAXRS

Tabla 8. Especificaciones de la configuración 2.

6.1.2. Pruebas e implementación

Se procede a definir las pruebas que se han realizado para la evaluación de rendimiento, así como la forma en la que cada una de las pruebas se ha implementado para poder llevarlas a cabo.

Pruebas generales

Este apartado engloba pruebas generales de funcionamiento. Estas pruebas están ligadas a la experiencia de usuario, por lo que aunque no son las más objetivas, son las que permiten evaluar la experiencia de uso. Las mediciones que se han realizado son las siguientes:

- Tiempo de arranque del ESB: con esta medición se pretende evaluar el tiempo de arranque del programa, desde que se ejecuta hasta que todos los *bundles* del sistema se cargan y ejecutan según proceda. Para ello, se utilizará la herramienta *top* del sistema Raspbian, que permite ver el tiempo de ejecución de los procesos desde que se inician. Se observará el tiempo llevado a cabo desde su arranque hasta que aparecen las primeras trazas del funcionamiento del controlador de dispositivos en la consola de administración de ServiceMix.
- Tiempo de despliegue: se hará una medición del tiempo que tarda un *bundle* en pasar a estado activo (y creado) desde que se despliega en caliente. Para ello, se desinstalará el *bundle* correspondiente al controlador de dispositivos, y se medirá el tiempo desde que se vuelve a instalar hasta que se muestra la primera traza en la consola.

Tiempo de registro

Esta prueba tiene como objetivo medir el tiempo que el controlador de dispositivos tarda en registrarse frente al sistema *e-GOTHAM*. De esta forma, y comparando los resultados con las dos configuraciones propuestas, se podrá evaluar si el tiempo empleado entra dentro del funcionamiento normal del sistema.

Para implementar esta prueba, se ha utilizado la clase *Date* de Java. El método *getTimeMillis()* permite obtener el tiempo en el cuál la clase *Date* fue instanciada, y por lo tanto se podrá guardar el instante que se quiera comparar. Además, este método permite obtener el tiempo como un entero, por lo cual una simple resta de dos de estos objetos que ejecuten este método permitirá saber el tiempo que se ha empleado.

Con la herramienta que provee la clase *Date* de Java necesaria para la medición de tiempo, Simplemente se deben instanciar dos de estos objetos. Los momentos en los que se instanciarán los métodos son los siguientes:

- Se instanciará un primer objeto del tipo *Date* en el momento justo de iniciación del controlador de dispositivos.
- El segundo objeto de instanciará en el instante inmediatamente posterior a la recepción del mensaje positivo de registro.

Por último, una simple resta como la que se ha definido, que se guardará en un fichero de medidas, permitirá observar el tiempo empleado.

Tiempos de petición de datos

La última prueba que se realizará tiene que ver con el tiempo empleado para que el sistema *e-GOTHAM* obtenga una medida pedida. La implementación de la prueba es la misma que la definida para el tiempo de registro. Para poder tener una aproximación más clara del rendimiento real, se han realizado dos pruebas independientes, que tienen que ver con la forma en que se genera la medida en el controlador de dispositivos. El número de muestras tomadas en cada una de las pruebas independientes varía, así como las configuraciones en las que se realiza.

- La primera prueba consiste en la devolución de un valor que se genera de forma aleatoria por parte del controlador de dispositivos.
- La segunda se basa en obtener un valor guardado en un fichero. Esta prueba permite ver el tiempo de acceso a un fichero.

La razón de incluir estas pruebas independientes es obtener más datos de comparación de ejecución entre una plataforma más potente, como un ordenador personal, y el *Raspberry Pi*.

6.2. Resultados y análisis parciales

En este apartado se muestran los resultados obtenidos de la realización de las pruebas, de los que se extraerán medidas estadísticas que permitirán generar una muestra del funcionamiento real del sistema. Dichas medidas estadísticas son las siguientes:

- Número de la muestra: determina el número de mediciones. Se representa con el símbolo *N*.
- Media: establece el valor medio obtenido de las mediciones. Sigue la fórmula 1:

$$\frac{\sum_{i=1}^{N} Ti}{N} \quad (1)$$

Siendo *Ti* el valor de cada una de las mediciones.

Los demás son parámetros propios de estadística, de los cuales no se muestran sus ecuaciones.

En cada una de las pruebas se contrapondrán los resultados de las diferentes configuraciones, y se hará una valoración previa en cada una de ellas.

Por último, se extraerán los datos totales obtenidos y se utilizarán las valoraciones previas para exponer los resultados finales, y así poder realizar una valoración.

Entre los datos finales que se extraerán, se evaluará el rendimiento del *Raspberry Pi* respecto al PC. No se trata de un rendimiento real, sino una definición para la comparación de tiempos. Este *rendimiento* sigue la se fórmula 2.

$$\mu = \frac{Tpc}{Trp} \quad (2)$$

Tpc: Tiempo medido en ms en el PC.

Trp: Tiempo medido en ms en el *Raspberry Pi*.

6.2.1. Pruebas generales

6.2.1.1. *Tiempo de arranque*

Para el tiempo de arranque del ESB, se han hecho diez mediciones separadas. Los resultados vienen dados por las tablas Tabla 9 y Tabla 10, y los gráficos Gráfico 1 y Gráfico 2.

Configuración 1. PC

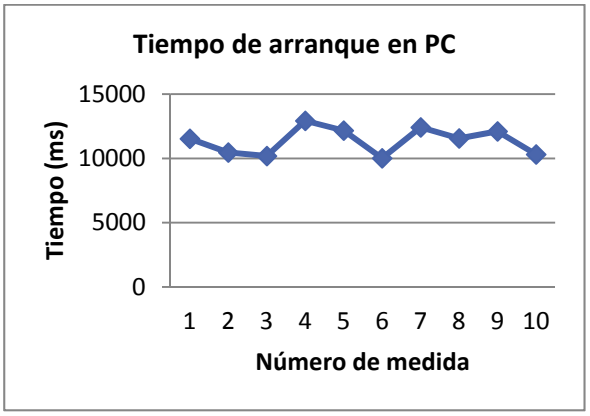

Gráfico 1. Medida de tiempo de arranque para Configuración 1. PC.

Datos estadísticos de tiempo de arranque (Configuración 1. PC)	
Número de la muestra (N)	10
Media o promedio	11366,9 ms
Desviación respecto de la media (Dm)	900,92
Desviación típica	1052,700068

Tabla 9. Datos estadísticos de tiempo de arranque en Configuración 1 PC.

Configuración 2. Raspberry Pi

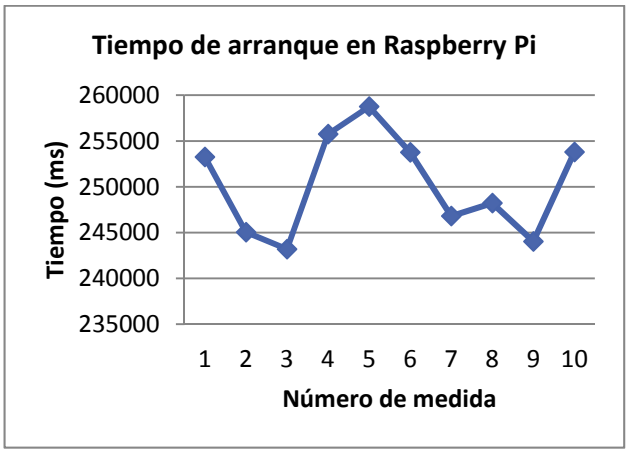

Gráfico 2. Medida de tiempo de arranque para Configuración 2. Raspberry Pi.

Datos estadísticos de tiempo de arranque (Configuración 2. *Raspberry Pi*)	
Número de la muestra (N)	10
Media o promedio	250259,8 ms
Desviación respecto de la media (Dm)	4801,6
Desviación típica	5456,707

Tabla 10. Datos estadísticos de tiempo de arranque en Configuración 2. Raspberry Pi.

Valoración de los resultados

Como se puede observar si se atiende al tiempo medio que tarda el *Raspberry Pi* en arrancar y cargar todos los *bundles* necesarios, se ve que este tiempo es exageradamente mayor que en el caso de un ordenador personal. También se puede observar que existe una desviación de unos 5 segundos respecto al valor medio de tiempo de arranque.

Estas cifras tan altas de tiempo para el *Raspberry Pi* indican una clara falta de memoria del sistema, que hace ralentizar el programa respecto al funcionamiento en un PC. Por otra parte, la CPU del dispositivo es de mucha menor velocidad, por lo que dificulta que se cargue con rapidez.

6.2.1.2. *Tiempo de despliegue*

Para el tiempo de despliegue, tiempo empleado en instalar y arrancar *bundle*, se han realizado 10 mediciones. Los resultados vienen dados por las tablas Tabla 11 y Tabla 12, y los gráficos Gráfico 3 y Gráfico 4.

Configuración 1. PC

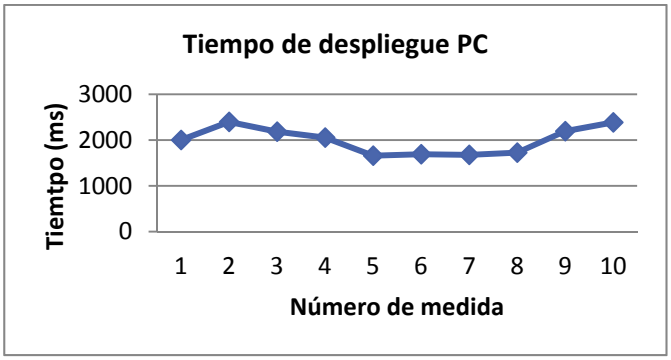

Gráfico 3. Medida de tiempo de despliegue para Configuración 1. PC.

Datos estadísticos de tiempo de despliegue (Configuración 1. PC)	
Número de la muestra (N)	10
Media o promedio	1997,8 ms
Desviación respecto de la media (Dm)	247,44
Desviación típica	293,1692

Tabla 11. Datos estadísticos de tiempo de despliegue en Configuración 1. PC.

Configuración 2. Raspberry Pi

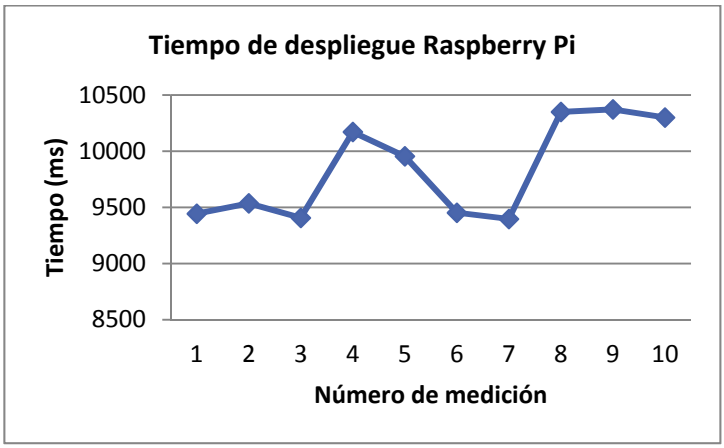

Gráfico 4. Medida de tiempo de despliegue para Configuración 2. Raspberry Pi.

Datos estadísticos de tiempo de despliegue (Configuración 2. *Raspberry Pi*)	
Número de la muestra (N)	10
Media o promedio	11366,9 ms
Desviación respecto de la media (Dm)	900,92
Desviación típica	1052,700068

Tabla 12. Datos estadísticos de tiempo de despliegue en Configuración 2. Raspberry Pi.

Valoración de los resultados

Al igual que en el caso anterior, el tiempo de despliegue vuelve a ser mucho mayor en el *Raspberry Pi* que en el ordenador personal. En este caso, puede deberse a una falta de velocidad de acceso al sistema de almacenamiento, ya que la interfaz para la tarjeta SD es mucho más lenta que la del disco duro del ordenador personal.

Si a todo esto se le suma la poca memoria RAM y la baja velocidad de procesamiento, se puede entender de nuevo la clara diferencia de tiempos promedio.

6.2.2. Registro

Para la medición del tiempo de registro, se han realizado 48 mediciones. Los resultados se muestran en las tablas Tabla 13 y Tabla 14, y los gráficos Gráfico 5 y Gráfico 6.

Configuración 1. PC

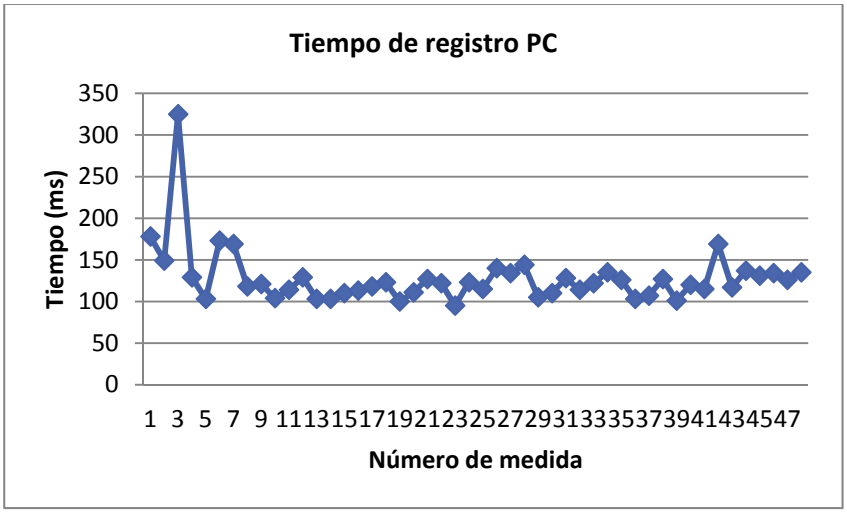

Gráfico 5. Medida de tiempo de registro para Configuración 1. PC.

Datos estadísticos de medición de tiempo de registro (Configuración 1. PC)	
Número de la muestra (N)	48
Media o promedio	128,2291667 ms
Desviación respecto de la media (Dm)	19,13888889
Desviación típica	34,79498593

Tabla 13. Datos estadísticos de medición de tiempo de registro en Configuración 1. PC.

Configuración 2. Raspberry Pi

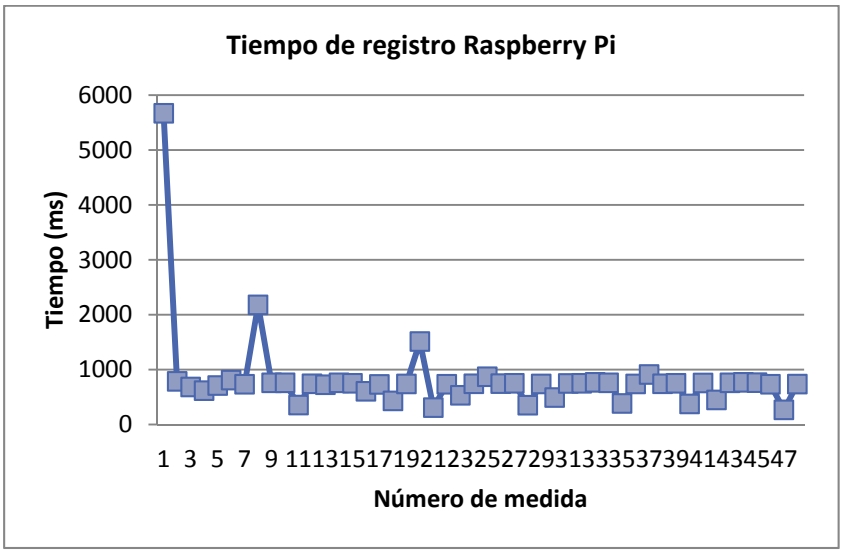

Gráfico 6. Medida de tiempo de registro para Configuración 2. Raspberry Pi.

Datos estadísticos de medición de tiempo de registro (Configuración 2. *Raspberry Pi*)	
Número de la muestra (N)	48
Media o promedio	822,7916667 ms
Desviación respecto de la media (Dm)	292,9600694
Desviación típica	771,8595661

Tabla 14. Datos estadísticos de medición de tiempo de registro en Configuración 2.

Valoración de los resultados

Se ve que, si bien en las anteriores pruebas los tiempos promedio de las dos configuraciones distaban mucho, ahora este tiempo se reduce considerablemente. Esto se debe a que el controlador de dispositivos apenas debe realizar operaciones computacionales para el registro, tan solo operaciones sobre la red para el envío de los mensajes. Por otra parte, el registro entero del dispositivo es realizado enteramente en el elemento del sistema *e-GOTHAM*, por lo que el tiempo que tarda en registrarse depende en su mayor parte de las capacidades de la máquina que sirva como soporte para dicho elemento.

Se puede observar que la desviación típica es mucho mayor en la configuración 2 que en la configuración 1. Esto se debe a un primer valor de medición muy alto en el *Raspberry Pi*, ya que se ejecuta el registro nada más haber iniciado y arrancado el ESB, y las conexiones y sesiones que se crean para las comunicaciones de red tardan más en crearse que en intentos posteriores.

6.2.3. Petición de datos

6.2.3.1. *Devolución de un valor aleatorio*

Para esta prueba, se han realizado 100 mediciones. Los resultados se muestran en las tablas Tabla 15 y Tabla 16, y los gráficos Gráfico 7 y Gráfico 8.

Configuración 1. PC

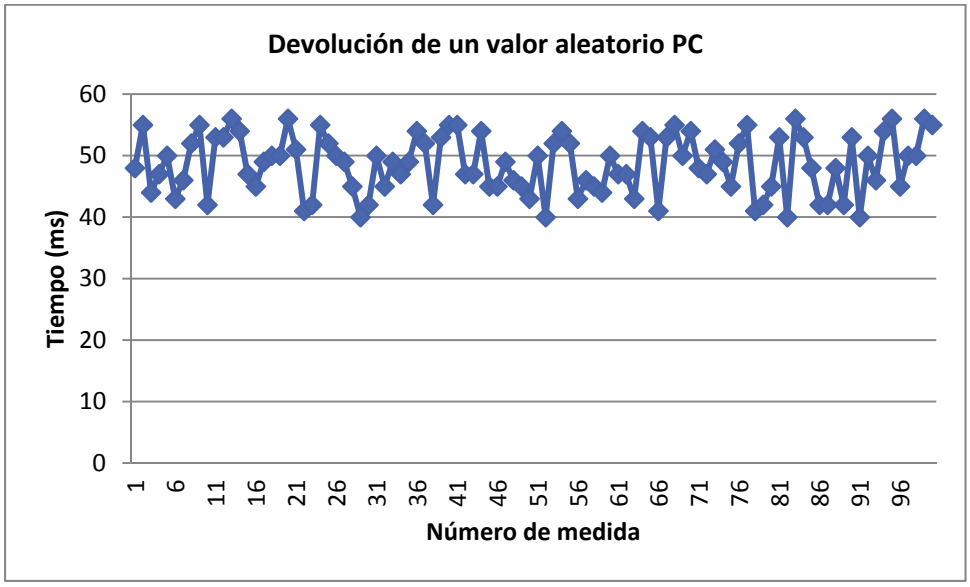

Gráfico 7. Medidas de tiempo de devolución de un valor aleatorio en Configuración 1. PC

Datos estadísticos de devolución de un valor aleatorio (Configuración 1. PC)	
Número de la muestra (N)	100
Media o promedio	48,61 ms
Desviación respecto de la media (Dm)	3,765625
Desviación típica	4,512984

Tabla 15. Datos estadísticos de devolución de un valor aleatorio en Configuración 1. PC.

Configuración 2. Raspberry Pi

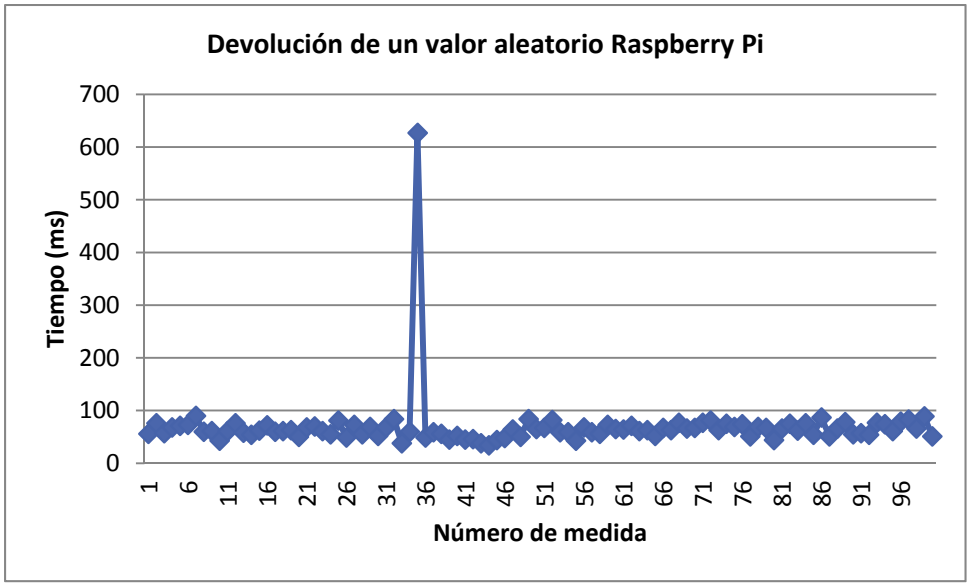

Gráfico 8. Medidas de tiempo de devolución de un valor aleatorio en Configuración 2. Raspberry Pi.

Datos estadísticos de devolución de un valor aleatorio (Configuración 2. *Raspberry Pi*)	
Número de la muestra (N)	100
Media o promedio	68,18 ms
Desviación respecto de la media (Dm)	10,1
Desviación típica	13,00640868

Tabla 16. Datos estadísticos de devolución de un valor aleatorio en Configuración 2. Raspberry Pi.

Valoración de los resultados

Si bien todas las pruebas anteriores daban una clara diferencia entre las medidas de tiempo en las dos configuraciones, en esta prueba se puede observar un claro acercamiento de los tiempos de las mediciones. No obstante, el rendimiento en el *Raspberry Pi* sigue siendo menor que en el ordenador, del orden de un 50% menor.

También las derivas del tiempo son de nuevo más estables en el ordenador personal que en el *Raspberry Pi.* De esta forma, mientras en la configuración 1 se consiguen derivas de un máximo de entorno a los 4 ms, en el *Raspberry Pi* este valor aumenta hasta los 10 ms aproximadamente.

Por último, se observa que el *Raspberry Pi* posee valores muy altos en momentos determinados de tiempo. Esto se debe a la sobrecarga del sistema a raíz de la ejecución del ESB, que a veces puede llegar a saturar el dispositivo hasta el punto de dar medidas de en torno a los 700 ms.

6.2.3.2. *Devolución de un valor leído desde un fichero*

Para esta prueba, se han realizado 100 mediciones. Los resultados se muestran en las tablas Tabla 17 y Tabla 18, y los gráficos Gráfico 9 y Gráfico 10.

Configuración 1. PC

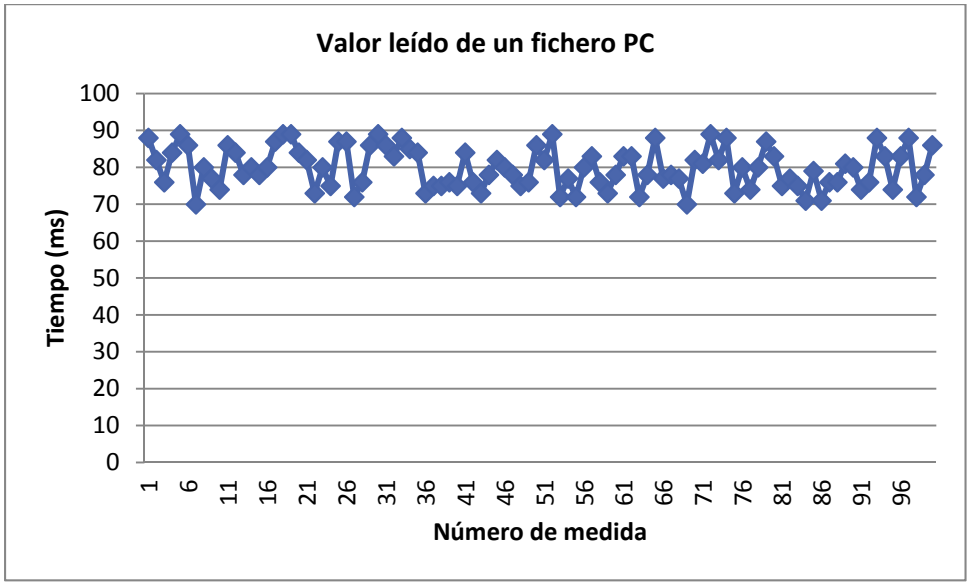

Gráfico 9. Medidas de tiempo de devolución de un valor leído de fichero en Configuración 1. PC.

Datos estadísticos de devolución de un valor leído de fichero (Configuración 1. PC)	
Número de la muestra (N)	100
Media o promedio	79,86 ms
Desviación respecto de la media (Dm)	4,779514
Desviación típica	5,469289

Tabla 17. Datos estadísticos de devolución de un valor leído de un fichero en Configuración 1.PC.

Configuración 2. Raspberry Pi

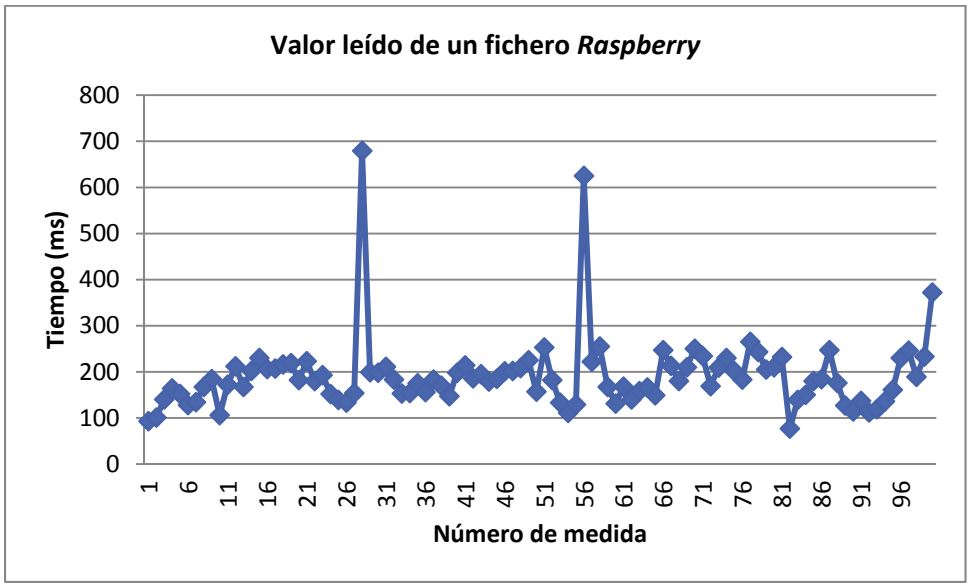

Gráfico 10. Medidas de tiempo de devolución de un valor fijo en Configuración 2. Raspberry Pi

Datos estadísticos de devolución de un valor aleatorio (Configuración 2. *Raspberry Pi*)	
Número de la muestra (N)	100
Media o promedio	191,13 ms
Desviación respecto de la media (Dm)	36,55903
Desviación típica	79,67029

Tabla 18. Datos estadísticos de devolución de un valor leído de un fichero en Configuración 2.Raspberry Pi.

Valoración de los resultados

De esta prueba se puede extraer una lectura clara, y es que el sistema de almacenamiento del *Raspberry Pi* es mucho más lento que el del ordenador personal, ya que está basado en tarjetas SD. De esta forma, la búsqueda de un fichero es más lenta.

Por otra parte, vemos que la interfaz I/O de Java para el manejo de ficheros se ejecuta de forma mucho más lenta en el *Raspberry Pi*, lo que provoca tiempos promedio mucho mayores.

6.3. Valoración final y toma de decisiones

En la Tabla 19 se engloban los resultados generales obtenidos, valorando el rendimiento obtenido.

	Configuración 1	Configuración 2	Rendimiento
Tiempo de arranque	11366,9	250259,8	0,0454
Tiempo de despliegue	1997,8	11366,9	0,1757
Tiempo de registro	128,2	822,8	0,1558
Devolución de valor aleatorio	48,61	68,18	0,7130
Devolución de valor leído de fichero	79,86	191,13	0,4178

Tabla 19. Datos finales de rendimiento.

- El tiempo de arranque en el PC es un 96% aproximadamente más rápido que en el *Raspberry Pi.*
- El tiempo de despliegue en el PC es un 87% aproximadamente más rápido que en el *Raspberry Pi.*
- El tiempo de registro en el PC es un 85% más rápido que en el *Raspberry Pi.*
- El tiempo de devolución de valores es tan sólo aproximadamente el doble en el *Raspberry Pi.*

Se puede observar que el rendimiento en las pruebas generales del *Raspberry Pi* es mucho menor que el obtenido en un PC. Esto es algo de esperar, debido a las diferencias en potencia computacional y en memoria.

No obstante, las mediciones de petición de datos arrojan datos muy satisfactorios, con tiempos tan sólo 50% (aproximadamente) más lentos en el *Raspberry Pi.*

El controlador de dispositivos realiza las operaciones de arranque y despliegue tan sólo una vez desde que se instala. Además, en un comportamiento adecuado del sistema, el dispositivo lógico sólo debe registrarse una vez. Por tanto, los valores que se pueden obtener en estas pruebas son mucho menos relevantes.

En cuanto el controlador central está activo y el controlador de dispositivos está corriendo, se pueden realizar las verdaderas operaciones que dan sentido al sistema, que se corresponden con la petición de datos. Se puede observar que los tiempos no sobrepasan la barrera de los 100 ms en el caso de datos provenientes de la lógica del controlador de dispositivos. La diferencia de rendimiento en los casos relevantes del sistema por tanto sería aproximadamente el doble en las dos configuraciones. Para evaluar finalmente esto, se tiene en cuenta que:

- El *Raspberry Pi* tiene un coste aproximadamente 10 veces menor que un ordenador con las características mencionadas.
- El consumo es también unas diez veces menor.

- El ordenador personal es difícil de ubicar, ya que requiere un gran espacio. En cambio el *Raspberry Pi* se puede llegar a ubicar incluso en algunos dispositivos físicos de gran tamaño.
- Las peticiones de datos del sistema se hacen de forma asíncrona. Por tanto, en ningún caso se puede producir un bloqueo del sistema.
- Los tiempos entre las peticiones de datos en un sistema como éste tienen un carácter periódico, pero no necesitan tener un carácter instantáneo. Por ello, los periodos de tiempo serán muy largos en comparación con el tiempo de devolución.

De estas consideraciones se extrae que un dispositivo 10 veces más rentable que un ordenador personal es tan sólo la mitad de lento, y que los tiempos de peticiones de datos reales están muy lejos de los umbrales máximos para el sistema.

A raíz de estas consideraciones, y teniendo en cuenta los resultados obtenidos, se puede concluir que el dispositivo *Raspberry Pi* sirve perfectamente como una herramienta para el despliegue de servicios dentro del sistema en el que se enmarca, pudiendo cumplir las funciones de un controlador local de dispositivos.

7. Conclusiones y trabajos futuros

Mediante el estudio del marco tecnológico del proyecto, se han podido sacar ciertas conclusiones sobre el estado del arte de las diferentes tecnologías usadas:

- Las *Smart Grid* están llamadas a ser la base para la construcción de los futuros sistemas de abastecimiento de energía, basados principalmente en las energías renovables.
- Las arquitecturas orientadas a servicios son un componente fundamental para el desarrollo de aplicaciones en el ámbito empresarial, y adquirirán una mayor relevancia si cabe en el futuro próximo.

En este proyecto se ha podido ver las diferentes soluciones para arquitecturas orientadas a servicios, basándose en los ESB actuales, de los cuales se ha podido concluir que:

- Existen numerosas soluciones, de las cuales las soluciones de código abierto son las más utilizadas. Además, son soluciones muy maduras y asentadas en el mercado tecnológico.
- La mayoría permiten una interoperabilidad muy alta, y trabajan en sistemas multiplataforma.
- Los ESB son las soluciones software para SOA más extendidas y con mejor funcionamiento.

Por otro lado, se ha conseguido demostrar que el *Raspberry Pi* es una efectiva herramienta de despliegue de servicios del sistema, con sus ventajas inherentes al ser un dispositivo de reducido tamaño y valor reducido. Esto se ha conseguido gracias a las pruebas realizadas sobre un controlador de dispositivos, que si bien funciona en un entorno que no será el final, no se trata de un sistema totalmente simulado, ya que permite recoger datos reales de un sensor de corriente. De este desarrollo se ha podido extraer algunas conclusiones:

- El desarrollo de servicios está cada vez más automatizado, y hay numerosas herramientas que facilitan la labor del desarrollador.
- La realización de pruebas de rendimiento basadas en tiempos de ejecución de operaciones proporcionan información con gran relevancia para la toma de decisiones de implementación de un sistema.

Del desarrollo general de este proyecto se han podido sacar también ciertas conclusiones subjetivas, como:

- La documentación existente para las arquitecturas orientadas a servicios es amplia, pero se encuentra poco centralizada y estructurada, por lo que dificulta el proceso de aprendizaje.
- El funcionamiento de los ESB debe mejorarse a lo largo de los siguientes años para ofrecer un mayor número de herramientas y una ejecución más coherente.
- La presencia de foros de información en Internet es muy amplia, y la mayoría de las veces es mucho más útil que la que podemos encontrar en documentaciones oficiales.
- La mayoría de libros dedicados al desarrollo de software distribuido en SOA son de difícil acceso, y una gran cantidad de ellos son demasiado generales, y con un coste desorbitado.
- Supone un gran éxito personal el haber podido llevar a cabo exitosamente el objetivo final de este proyecto, un objetivo que a priori parecía difícil de poder conseguir.

Gracias a la infraestructura de *e-GOTHAM* he se ha podido desarrollar el controlador de dispositivos. Agradezco al grupo de investigación GRyS el poder haber utilizado dicha infraestructura, y también por la ayuda prestada a la hora de enseñarme su funcionamiento.

7.1. Trabajos Futuros

Este proyecto, al estar englobado en un sistema mucho mayor, y aún en desarrollo, tiene múltiples líneas de mejora, e incluso se pueden apreciar muchas nuevas líneas de investigación que lleven al desarrollo de nuevas tecnologías dentro del marco en el que se mueve.

Mejoras de funcionamiento del *Raspberry Pi*

Para poder mejorar el rendimiento del *Raspberry Pi* como controlador local, se pueden realizar ciertas acciones que deberán ser evaluadas y tenidas en cuenta, como:

- Utilización de *overclock* del procesador, que permita tener una capacidad mayor de procesamiento.
- Exploración de nuevos sistemas operativos que surjan con el paso del tiempo.
- Actualización del ESB que permita obtener versiones mejoradas y con mayor rendimiento de los componentes instalados en él.
- Utilización de redes inalámbricas para su interconexión con otros dispositivos, que permita una mayor ubicuidad del dispositivo.

Si bien las funcionalidades del controlador de dispositivos son las exigidas para el sistema, se proponen algunas modificaciones que permitan tener un control mayor sobre su funcionamiento, y los dispositivos administrados.

- Implementar métodos de *timeout* y reconexión para evitar bloquear el funcionamiento del controlador de dispositivos frente a posibles errores como:
    - Conexiones y sesiones erróneas o cerradas.
    - Transmisión y recepción de mensajes fallida.
- Creación de herramientas para el desarrollador, que permitan depurar el funcionamiento del dispositivo lógico.
- Carga de propiedades dinámica para el entorno de funcionamiento.
- Supervisión de los algoritmos actuales y depuración para un aumento del rendimiento.
- Inclusión de la información de contexto de los dispositivos.
- Obtención dinámica de los datos de los dispositivos físicos.

La implementación realizada no cubre todas las funcionalidades previstas en la arquitectura *e-GOTHAM*, por lo que se podrán lógicamente incorporar más funcionalidades previstas, como las que se citan relacionadas con las comunicaciones entre los diversos elementos de la arquitectura:

- Utilización de AMQP.
- Extensión de comunicaciones OSGi entre los ESB del sistema, para disponer de un sistema OSGi totalmente distribuido.

- Creación de nuevas interfaces REST.

Una ampliación necesaria para el desarrollo del sistema será la inclusión de dispositivos no sólo medidores, si no que se pueda interactuar con ellos para realizar operaciones. Se deben incluir nuevos dispositivos como:

- Actuadores
- Nuevas redes de sensores
- Inclusión de dispositivos legados.

Otra importante ampliación es la de estudiar la inclusión de servicios adicionales de Smart Home integrados con los de *Smart Grid*, en el *Raspberry Pi*.

8. Bibliografía

[1] L. Melo B. y N. Espinosa, «Ineficiencia en la distribución de energía eléctrica.,» Diciembre 2004. [En línea]. Available: http://www.banrep.gov.co/docum/ftp/borra321.pdf. [Último acceso: Febrero 2014].

[2] R. Córdoba Hernández, «Energía: consumo, contaminación y cambio climático,» 2004. [En línea]. Available: http://habitat.aq.upm.es/boletin/n34/arcor_3.html. [Último acceso: Febrero 2014].

[3] SMARTGRID.gov, «What is the Smart Grid?,» [En línea]. Available: https://www.smartgrid.gov/the_smart_grid. [Último acceso: Febrero 2014].

[4] Red eléctrica de España, «¿Qué son las Smartgrid?,» [En línea]. Available: http://www.ree.es/es/red21/redes-inteligentes/que-son-las-smartgrid.

[5] Siemens, «Siemens Smart Grid,» [En línea]. Available: http://w3.siemens.com/smartgrid/global/en/pages/default.aspx. [Último acceso: Febrero 2014].

[6] Hewlett Packard, «Smart Grid solutions,» [En línea]. Available: http://www8.hp.com/es/es/business-services/it-services.html?compURI=1079607#.U7EIZ_l_uQA.

[7] C. Felix Covrig, M. Ardelean, J. Vasiljevska, A. Mengolini, G. Fulli y E. Amoiralis, «Smart Grid Projects Outlook 2014,» Joint Research Centre of European Comission, 2014.

[8] e-Gotham, «e-Gotham Smart grid Objectives,» [En línea]. Available: http://www.e-gotham.eu/index.php/e-gothamproj/objectives. [Último acceso: Febrero 2014].

[9] I3RES, «ICT-based intelligent integration of RES,» [En línea]. Available: http://www.i3res.eu/v1/. [Último acceso: Febrero 2014].

[10] T. Erls, Service-Oriented Architecture. Concepts, Technology and Design., Prentice Hall, 2005.

[11] SOA Blueprint, «SOA Practitioner's Guide Part 2. SOA Reference Architecture.,» [En línea]. Available: http://www.soablueprint.com/whitepapers/SOAPGPart2.htm. [Último acceso: Febrero 2014].

[12] D. Krafzig, K. Banke, D. Slama y F. Lindner, Enterprise SOA, Prentice Hall, 2005.

[13] OSGi Alliance, «About the OSGi Alliance,» [En línea]. Available: http://www.osgi.org/About/HomePage. [Último acceso: Febrero 2014].

[14] OSGi Alliance, «OSGi Architecture,» [En línea]. Available: http://www.osgi.org/Technology/WhatIsOSGi. [Último acceso: Febrero 2014].

[15] R. Montero, «Java Hispano, OSGI,» [En línea]. Available: http://www.javahispano.org/storage/contenidos/OSGI_Roberto_Montero.pdf. [Último acceso: Febrero 2014].

[16] Spring source, «Spring Dynamic Modules Reference Guide,» [En línea]. Available: http://docs.spring.io/osgi/docs/1.2.1/reference/html/. [Último acceso: Marzo 2014].

[17] Apache Ares, «Apache Aries Blueprint,» [En línea]. Available: http://aries.apache.org/modules/blueprint.html. [Último acceso: Marzo 2014].

[18] OSGi Alliance, «Benefits of using OSGi,» [En línea]. Available: http://www.osgi.org/Technology/WhyOSGi. [Último acceso: Junio 2014].

[19] OSGi Alliance, «OSGi Alliance Specifications,» [En línea]. Available: http://www.osgi.org/Specifications/HomePage. [Último acceso: Marzo 2014].

[20] Berkeley, «What is MOM?,» [En línea]. Available: http://courses.ischool.berkeley.edu/i206/f97/GroupB/mom/what_is_mom.html. [Último acceso: Marzo 2014].

[21] Banda base, «¿Qué es el middleware orientado a mensajes?,» [En línea]. Available: http://bandabase.com/que-es-el-middleware-orientado-a-mensajes/. [Último acceso: Marzo 2014].

[22] Universidad de Alicante, «Experto Java, MOM,» [En línea]. Available: http://www.jtech.ua.es/j2ee/restringido/comp-ee/sesion05-apuntes.html. [Último acceso: Marzo 2014].

[23] Oracle, «Jave EE 6 Tutorial,» [En línea]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html. [Último acceso: Marzo 2014].

[24] M. Richards, «Understanding the differences between AMQP & JMS,» 2011. [En línea]. Available: http://www.wmrichards.com/amqp.pdf. [Último acceso: Marzo 2014].

[25] Microsoft, «Introducción al protocolo AMQP del Service Bus,» [En línea]. Available: http://msdn.microsoft.com/es-es/library/jj841072.aspx. [Último acceso: Marzo 2014].

[26] RabbitMQ, «RabbitMQ official page,» [En línea]. Available: http://www.rabbitmq.com/. [Último acceso: Marzo 2014].

[27] Apache, «Apache ActiveMQ,» [En línea]. Available: http://activemq.apache.org/. [Último acceso: Marzo 2014].

[28] W3C, «Guía breve de Servicios Web,» [En línea]. Available: http://w3c.es/Divulgacion/GuiasBreves/ServiciosWeb . [Último acceso: Marzo 2014].

[29] A. Sur, «Restful Crud Operation on a WCF Service,» 3 Octubre 2010. [En línea]. Available: http://www.codeproject.com/Articles/115054/Restful-Crud-Operation-on-a-WCF-Service. [Último acceso: Abril 2014].

[30] ICS, «Chapter 5 - Representional State Transfer (REST),» [En línea]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Último acceso: Marzo 2014].

[31] Oracle, «SOA - Enterprise Service Bus,» Julio 2013. [En línea]. Available: http://www.oracle.com/technetwork/articles/soa/ind-soa-esb-1967705.html. [Último acceso: Junio 2014].

[32] A. Gómez-Goiri, M. Emaldi Manrique y D. López de Ipiña, «Middleware semántico orientado a recursos para entornos ubicuos,» *Novatica,* vol. 1, nº 209, p. 16, 2011.

[33] V. Hernández Díaz, G. Rubio Cifuentes y J.-F. Martínez, «Semántica para Smart Cities - Asignatura Aplicaciones Telemáticas Avanzadas,» ETSIST, Madrid, 2014.

[34] Broadcom, «BCM2835,» [En línea]. Available: http://www.broadcom.com/products/BCM2835. [Último acceso: Marzo 2014].

[35] Raspberry Pi Foundation, «Raspberry Pi Foundation official page,» [En línea]. Available: http://www.raspberrypi.org/. [Último acceso: Enero 2014].

[36] J. Vicente Vallejo, Proyecto Fin de Carrera. Gestión del conocimiento en redes de sensores inalámbricos, Madrid: ETSIS de Telecomunicación. UPM, 2014.

[37] Arduino, «Sede web de Arduino,» [En línea]. Available: http://www.arduino.cc/. [Último acceso: Junio 2014].

[38] Apache ServiceMix, «Apache Servicemix official page,» [En línea]. Available: http://servicemix.apache.org/. [Último acceso: Marzo 2014].

[39] Fusesource, «Fuse ESB,» [En línea]. Available: http://fuse.fusesource.org/. [Último acceso: Marzo 2014].

[40] Red Hat, «JBoss Fuse,» [En línea]. Available: http://www.jboss.org/products/fuse/overview/. [Último acceso: Marzo 2014].

[41] MuleSoft, «Mule ESB,» [En línea]. Available: http://www.mulesoft.org/what-mule-esb. [Último acceso: Marzo 2014].

[42] Open ESB, «Open ESB official site,» [En línea]. Available: http://www.open-esb.net/. [Último acceso: Abril 2014].

[43] Petals ESB, «Petals ESB Features,» [En línea]. Available: http://petals.ow2.org/features.html.

[44] WSO2, «WSO2 ESB,» [En línea]. Available: http://wso2.com/products/enterprise-service-bus/. [Último acceso: Abril 2014].

[45] Adroit Logic, «Ultra ESB,» [En línea]. Available: http://adroitlogic.org/products/ultraesb.html. [Último acceso: Febrero 2014].

[46] Talend, «Talend ESB,» [En línea]. Available: http://www.talend.com/products/esb. [Último acceso: Febrero 2014].

[47] Membrane, «Membrane Service Proxy,» [En línea]. Available: http://www.membrane-soa.org/service-proxy/. [Último acceso: Marzo 2014].

[48] IBM, «WebSphere ESB,» [En línea]. Available: http://www-03.ibm.com/software/products/es/wsesb. [Último acceso: Abril 2014].

[49] Oracle, «Oracle ESB,» [En línea]. Available: http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html. [Último acceso: Abril 2014].

[50] Microsoft, «BizTalk Server,» Abril 2014. [En línea]. Available: http://www.microsoft.com/es-es/biztalk/default.aspx.

[51] Apache Maven, «Apache Maven home page,» [En línea]. Available: http://maven.apache.org/guides/introduction/introduction-to-the-pom.html. [Último acceso: Marzo 2014].

[52] Apache Maven, «Introduction to the POM,» [En línea]. Available: http://maven.apache.org/guides/introduction/introduction-to-the-pom.html. [Último acceso: Marzo 2014].

[53] Apache Maven, «Introduction to Archetypes,» [En línea]. Available: http://maven.apache.org/guides/introduction/introduction-to-archetypes.html. [Último acceso: Abril 2014].